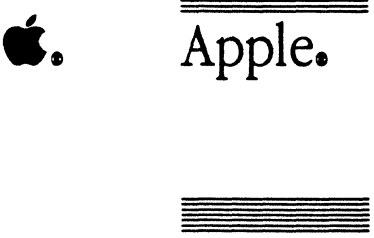


Apple. AppleTalk. for VMS™

Release Note for
Version 2.0

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or in part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© Apple Computer, Inc., 1988
20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Apple, the Apple logo, and AppleTalk are registered trademarks of Apple Computer, Inc.

Ethernet is a registered trademark of Xerox Corporation.

VAX, VMS, and DECnet are trademarks of Digital Equipment Corporation.

ITC Avant Garde Gothic, ITC Garamond, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated.

Varityper is a registered trademark, and VT600 is a trademark, of AM International, Inc.

Simultaneously published in the United States and Canada.

Contents

Preface **About This Release Note vii**

Intended audience vii
Structure of this document vii
Associated documents viii
Conventions used in this document viii

Part 1 **Introduction, Setup, and Operation 1**

Chapter 1 **Introduction to Version 2.0 3**

Overview of version 2.0 4
New and changed features 5
New features 5
Changed features 5

Chapter 2 **AppleTalk for VMS Installation 7**

Overview 8
Preparing for installation 8
Running the installation procedure 11

Chapter 3 **AppleTalk for VMS Updating 19**

Overview 20
Preparing to update 20
Running the update procedure 23

Chapter 4 AppleTalk for VMS Operation 31

Overview 32
The startup procedure 32
Multiple virtual networks 33
Restarting the Bridge Process 33
Shutdown 34
    Shutting down all processes 34
    Shutting down a single process 34

Part 2 AppleTalk Data Stream Protocol 35

Chapter 5 Using ADSP 37

Overview 38
General features of ADSP 38
The VMS implementation of ADSP 38
Opening and closing a connection 39
Sending and receiving data bytes 40
    Breaking the data stream into logical messages 40
Sending and receiving attention messages 41
Forward-resetting the connection 41
Implementing connection listeners 41
    Establishing connection listeners 42
    Receiving, accepting, and denying connection requests 43
Obtaining connection status 43
Tuning ADSP 44
Breaking a connection 44
Observing ADSP's internal significant events 45
Synchronizing client operations with ADSP 46
Buffer management 47
    Buffer management internals 48
Using tagged lists 49

Chapter 6 ADSP Call Formats 51

ATK$DSP_ACCEPT—Accept a connection with a remote end 53
ATK$DSP_CLEAR_COUNTERS—Clear the ADSP internal counters 59
ATK$DSP_CLOSE—Close a connection 60
ATK$DSP_CREATE_CON_END—Create a connection end 63
ATK$DSP_CREATE_LIS_SOC—Create a connection-listening socket 65
ATK$DSP_DELETE_LIS_SOC—Delete a connection-listening socket 66
ATK$DSP_DENY_CON_REQ—Deny a connection request 67
ATK$DSP_ESTABLISH—Establish a connection with a remote end 69
ATK$DSP_FLUSH—Flush the send queue 73

ATK$DSP_FORWARD_RESET—Forward-reset the connection 74
ATK$DSP_GET_BUF—Get a transmission buffer 76
ATK$DSP_GET_BYTE—Get a single byte 77
ATK$DSP_GET_CON_REQ—Get a connection request 78
ATK$DSP_GET_COUNTERS—Get ADSP internal counters 81
ATK$DSP_GET_DEFAULTS—Get ADSP default operating parameters 82
ATK$DSP_GET_STATUS—Get connection status 83
ATK$DSP_LISTEN—Passively open a connection 84
ATK$DSP_OPEN—Actively open a connection 89
ATK$DSP_PUT_BYTE—Send a single byte 94
ATK$DSP_READ—Read data from a connection 95
ATK$DSP_RECV_ATTENTION—Receive an attention message 98
ATK$DSP_RET_BUF—Return a receive buffer 101
ATK$DSP_SEND_ATTENTION—Send an attention message 102
ATK$DSP_SEND_EOM—Send an end-of-message marker 105
ATK$DSP_SET_DEFAULTS—Set ADSP default operating parameters 106
ATK$DSP_WRITE—Write data to a connection 107

Appendix A Tuning Parameter Identifiers 109

ATK$K_ATTNBCKOFF 110
ATK$K_ATTNINTRVL 110
ATK$K_FWDRSTBCKOFF 110
ATK$K_FWDRSTINTRVL 111
ATK$K_OPENINTRVL 111
ATK$K_OPENMAXRETRY 111
ATK$K_RCVQCPYTHRESH 112
ATK$K_RCVQMAXSIZE 112
ATK$K_RESTARTBCKOFF 112
ATK$K_RESTARTINTRVL 113
ATK$K_RETRANSBCKOFF 113
ATK$K_RETRANSINTRVL 113
ATK$K_RETRANSTHRESH 114
ATK$K_SNDQBUFSIZE 114
ATK$K_SNDWDWCLOBCKOFF 114
ATK$K_SNDWDWCLOINTRVL 115
ATK$K_SNDWDWCLOTHRESH 115
ATK$K_TRANSTHRESH 115
ATK$K_TRANSTIMERINTRVL 116

Appendix B Connection Status Identifiers 117

Appendix C ADSP Internal Counter Identifiers 119

Preface

About This Release Note

This release note describes new and changed features of AppleTalk® for VMS™, version 2.0, which supersedes version 1.6. This note replaces the *AppleTalk for VMS Installation and Operation Guide* issued with version 1.6. In addition, this note supplements the *AppleTalk for VMS Protocol Support Library Reference Manual.*

Intended audience

This manual is intended for programmers who use AppleTalk for VMS services in software development, as well as for system managers who install the AppleTalk for VMS programs. The manual assumes that you are familiar with the VAX/VMS operating system, with data communication principles, and with the VAX/VMS documentation set available from Digital Equipment Corporation (Digital).

Structure of this document

This manual has two parts, both subdivided into chapters.

- Part 1 introduces version 2.0 of the AppleTalk for VMS software, and shows how to install, update, start, and stop the software.
- Part 2 describes the AppleTalk for VMS client interface to the AppleTalk Data Stream Protocol (ADSP).

Associated documents

The following manuals from Apple Computer, Inc., document the AppleTalk network architecture, the interface provided to client processes of AppleTalk, the architecture of the AppleTalk for VMS system, and the services and interface it provides to client processes.

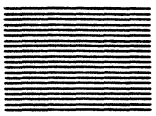
- *Inside AppleTalk* (Apple Programmer's and Developer's Association)
- *Inside Macintosh,* Volume II, Chapter 10, "The AppleTalk Manager"
- *Inside Macintosh,* Volume IV, Chapter 23, "The AppleTalk Manager"
- *Inside Macintosh,* Volume V, Chapter 28, "The AppleTalk Manager"
- *AppleTalk for VMS Architecture and Implementation*
- *AppleTalk for VMS Protocol Support Library Reference Manual*
- *AppleTalk for VMS Bridge Control Program Guide*

The VAX™/VMS™ documentation set, available from Digital Equipment Corporation, documents the use of VAX/VMS system services and software installation procedures.

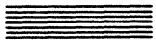
Conventions used in this document

Convention	Meaning
`file-spec,...`	Horizontal ellipsis indicates that *additional* parameters, values, or information can be entered or supplied
`[logical-name]`	Square brackets indicate that the enclosed item is *optional* (except when these brackets are used in a file name; for example, `DUA0:[SYS0.SYSMGR]PORTDESC.TEXT`). When an optional parameter is omitted from a call, a comma must mark the omitted parameter, unless the parameter is the final one of the call. For example, two optional parameters have been omitted from the following call: `ATK$DSP_Close(Refnum, Wait, , , ASTHandler, ASTParam)`

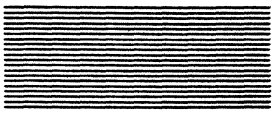
Other conventions used in this document are described in the VAX/VMS documentation set, available from Digital Equipment Corporation.



Part I



Introduction, Setup, and Operation



Chapter 1

Introduction to Version 2.0

Overview of Version 2.0

Version 2.0 of the AppleTalk® for VMS™ product consists of this release note, a magnetic tape or tape cartridge storing the AppleTalk for VMS software, and the following documents:

- *AppleTalk for VMS Architecture and Implementation*
- *AppleTalk for VMS Protocol Support Library Reference Manual*
- *AppleTalk for VMS Bridge Control Program Guide*

Note that this release note replaces the *AppleTalk for VMS Installation and Operation Guide* issued with version 1.6 of AppleTalk for VMS.

The AppleTalk for VMS software consists of the following programs:

- Virtual AppleTalk Network Driver
- AppleTalk Bridge Process
- AppleTalk for VMS Library
- AppleTalk Bridge Control Program

The first three of these programs are the logical tools that allow the AppleTalk network architecture to be implemented in the VAX/VMS environment. They contain a wide range of executable VMS procedures that perform AppleTalk protocol services, as well as the processes and device driver necessary to implement network operation. The fourth program, the AppleTalk Bridge Control Program, manages the AppleTalk Bridge Process. For more information about these programs, see the appropriate manual in this distribution kit.

New and changed features

This section lists differences between version 2.0 of AppleTalk for VMS and version 1.6 of the product.

New features

The following features are new for version 2.0:

- The Protocol Support Library includes the AppleTalk Data Stream Protocol (ADSP).
- AppleTalk for VMS supports versions 4.6 to 5.0 of the VMS operating system.

Changed features

Version 2.0 of AppleTalk for VMS incorporates the following changes:

- optimizes certain routines of the Protocol Support Library
- corrects problems found in earlier versions of the Bridge Process, the Virtual Network Driver, and the Protocol Support Library
- does not support versions of the VMS operating system earlier than 4.6 or versions of the VAX C run-time library earlier than 2.1
- does not include the AppleTalk for VMS Examples found in version 1.6, and does not support these examples

Chapter 2

AppleTalk for VMS Installation

Overview

This chapter explains how to install AppleTalk for VMS for the first time. For instructions on updating AppleTalk for VMS to version 2.0, see Chapter 3. AppleTalk for VMS installation consists of two main steps: preparing for installation, and running the AppleTalk for VMS installation procedure. The rest of this chapter explains how to perform these two main steps.

Important

AppleTalk for VMS should be installed only by the system manager. The system manager should be familiar with VMS software installation procedures, hidden devices (rooted directories), directory and file protection, Digital Command Language (DCL), and a text editor such as Digital's EDT software.

Preparing for installation

Before installing AppleTalk for VMS, you should prepare your system by following the steps listed in this section. After completing these steps, you will be ready to run the AppleTalk for VMS installation procedure.

To prepare your system for installation:

1. **Log in to the SYSTEM account of your VAX computer.**
2. **Make sure the VAX computer is running the following required software:**

   VMS operating system, version 4.6 or later

   VAX C run-time library, version 2.1 or later

Important

Make sure you have installed all software patches for the version of the VMS operating system you are running. Otherwise, AppleTalk for VMS will not operate properly. Installing all patches is especially important if you are running version 5.0 of VMS. For copies of VMS software patches, contact Digital Equipment Corporation.

---

3. **Make sure the SYSTEM account meets the following requirements:**

   The user name is SYSTEM.

   The user identification code (UIC) is [1,4].

   The account has access to all system privileges.

   All system privileges are enabled.

4. **Make sure there are enough free blocks of storage on your system disk.**

   AppleTalk for VMS requires 1500 blocks termporarily during installation; it requires 200 blocks permanently on the system disk.

5. **Back up your system disk.**

6. **Decide the name of the rooted directory where you will store the AppleTalk for VMS software.**

   Note that a rooted directory name must contain a trailing period (.)—for example, `DUA0:[SYS0.]` is a valid name for a rooted directory.

7. **Decide the password and user identification code (UIC) of the account that the installation procedure creates for AppleTalk for VMS.**

   The installation procedure creates a VMS account for AppleTalk for VMS; the user name of this account is `APPLETALK`. The UIC that you choose for this account must be unique among all accounts on your system. The installation procedure does not make sure that the UIC is unique. Note that UICs are octal numbers (base 8).

8. **Prepare to set up the AppleTalk for VMS Bridge.**

   You prepare to set up the AppleTalk for VMS Bridge Process by creating a port descriptor file. (For a description of the Bridge Process, see the *AppleTalk for VMS Architecture and Implementation.*) A port descriptor file is a text file storing one or more port descriptor lines. When the Bridge Process initializes, it uses the port descriptor file to set up the port handlers. The AppleTalk for VMS installation procedure creates the port descriptor file for you, using information you supply during the procedure. The port descriptor file is stored in the directory `ATK$ROOT:[APPLETALK]`.

❖ *Note:* For information regarding parameters and descriptions of the different port handlers supplied with AppleTalk for VMS, refer to the *AppleTalk for VMS Protocol Support Library Reference Manual,* Appendix A.

Gather the following information for each port:

- the `Port Driver File Name`, the file name of a port driver shareable image (only the file name portion of the file specification is used)
- the `Port Information String`, a port-dependent information string that is optional for some port types.
- the `Port Name`, a user-specified string of ASCII characters that uniquely identifies a particular port.
- the `Zone Name`, the AppleTalk zone name being assigned to the port's network. The `Zone Name` is optional, and is valid only for ports connected to AppleTalk networks. However, when omitting the `Zone Name` but supplying a port information string, you must supply a colon to mark the omitted `Zone Name`.

Although the installation procedure creates the port descriptor file for you, you can edit the file at any time after the procedure completes. However, the changes you make take effect only after you stop and restart the Bridge Process. The brief description that follows will help you edit the port descriptor file.

The port descriptor file consist of port descriptor lines, of which there are two types: `INIT` and `OPEN`. The `INIT` line initializes the specified port handler; each port handler must be initialized before it can be opened. The `OPEN` line opens the specified port handler.

The format of the `INIT` line is

```
INIT:Port Driver File Name:Port Information String
```

The format of the `OPEN` line is

```
OPEN:Port Name:Port Driver File Name:Zone Name:Port
Information String
```

When gathering information for the port descriptor file, carefully consider the network and port assignments. For the virtual AppleTalk network to be accessible to other AppleTalk networks, the port descriptor file must have at least four port descriptor lines. For example, on a VAX using Ethernet® to connect to other AppleTalk bridges, the port descriptor file must contain at least four port descriptor lines:

```
INIT:ATKVLAP
INIT:ATKAELAP
OPEN:VLAP Port:ATKVLAP:Zone Name:Network Number
OPEN:AELAP Port:ATKAELAP
```

The `VLAP` port is used to link to the virtual AppleTalk network inside the VAX computer. `Zone Name` is the name of the zone you wish to assign to the virtual AppleTalk network, and `Network Number` is the hexadecimal number that you wish to assign to the virtual AppleTalk network. The network number must be unique in your AppleTalk internetwork.

The `AELAP` port enables the Bridge Process to communicate with other bridges on the Ethernet.

9. **Check that any communication hardware to be used with AppleTalk for VMS is working.**
10. **Make sure the batch queue `SYS$BATCH` is operating and not stopped.**

    ```
    $ SHOW QUEUE SYS$BATCH
    ```

    If this queue is stopped or not operating, AppleTalk for VMS will not start.
11. **Set the default directory to `SYS$UPDATE`.**

    ```
    $ SET DEFAULT SYS$UPDATE
    ```
12. **Load the AppleTalk for VMS version 2.0 distribution tape on a tape drive, but do not mount the tape.**

Running the installation procedure

After preparing your system for installation as described in the previous section, you run the AppleTalk for VMS installation procedure. The installation procedure is started by the `VMSINSTAL` command procedure (`VMSINSTAL.COM`), part of the VMS operating system. For general information about using `VMSINSTAL` or about installing VMS layered products, see the VAX/VMS documentation set, available from Digital Equipment Corporation.

The instructions in this section assume that your kit was distributed on magnetic tape and that you will use device `MSA0:`. If you are using a different distribution medium or a different device, substitute the name of the medium or device you are using.

❖ *Note:* All installations of AppleTalk for VMS software are recorded in the log file `ATK$ROOT:[APPLETALK]APPLETALK.INS_LOG`. You can examine this file to see a history of AppleTalk for VMS installations.

To run the AppleTalk for VMS installation procedure, perform the following steps:

1. **Start the installation command procedure by entering the following DCL command at the VMS system prompt ($):**

```
$ @sys$update:vmsinstal atkvms020 msa0:
```

   The installation procedure displays several lines that look like these:

```
  VAX/VMS Software Product Installation Procedure V5.0


It is 2-SEP-1988 at 14:39.
Enter a question mark (?) at any time for help.

%VMSINSTAL-W-DECNET, Your DECnet network is up and running.
%VMSINSTAL-W-ACTIVE, The following processes are still active:
        _OPA0:
* Do you want to continue anyway [NO]?
```

2. **Enter YES.** (Except where noted in these instructions, entering NO causes the installation procedure to terminate, returning you to the VMS system prompt.)

   The installation procedure displays:

```
* Are you satisfied with the backup of your system disk [YES]?
```

3. **Enter YES.**

The installation procedure displays a question like this one:

```
Please mount the first volume of the set on  MSA0:.
* Are you ready?
```

4. **Enter YES after you have mounted the volume.**

The installation procedure displays several lines that look like these:

```
%MOUNT-I-MOUNTED, ATKVMS mounted on _$1$MSA0: (CALVIN)

The following products will be processed:

  ATKVMS V2.0


        Beginning installation of ATKVMS V2.0 at 14:40

%VMSINSTAL-I-RESTORE, Restoring product saveset A ...

        AppleTalk for VMS, Version V02-00
        A product of Apple Computer Inc.

        This installation installs the following applications:

        AppleTalk for VMS Library              Version: V02-01A
        AppleTalk Bridge Process               Version: V02-02A
        Virtual AppleTalk Network Driver       Version: V02-002
        AppleTalk Bridge Control Program       Version: V02-01A

* Please enter your name:
```

5. **Enter your name.**

The installation procedure displays the following question:

```
* Do you want to purge files replaced by this installation [YES]?
```

6. **Enter YES.** (If you enter NO, the installation procedure continues and does not purge the files.)

The installation procedure displays the following text:

```
AppleTalk for VMS  requires a  system-wide logical  ATK$ROOT to be
assigned to a rooted directory.  ATK$ROOT is used as the top-level
directory for any  other directories  that  are created  on behalf of
the AppleTalk for VMS product.  The next question  asks for the full
name of the  rooted directory to define as ATK$ROOT.  If  you specify a
rooted  directory  name that does not already exist, it will be created
for you.  The  default rooted  directory  used is SYS$SPECIFIC:.  Do
NOT  include [000000]  as part of a  directory specification.  Remember,
a rooted  directory name  must contain a trailing  period (.),  such  as
DUA0:[SYS0.].  Although  you  may specify a logical name,  it must
translate to  a rooted directory name (i.e. directory name with trailing
period).

* Rooted directory name:
```

7. **Enter the full name of the rooted directory—for example, `DUA0:[SYS0.]`.** Do not include `[000000]` when entering the name.

The installation procedure displays lines like these:

```
      ATK$ROOT will be assigned to the following rooted directory:
      DUA0:[SYS0.]

* Is this correct [Y]?
```

8. **Enter YES to continue.** (Entering NO enables you to reenter the directory name.)

The installation procedure displays a message and text that look like this:

```
%CREATE-I-EXISTS, DUA0:[SYS0] already exists

AppleTalk Networking Background Information
---------------------------------------------------

AppleTalk networks  can be  expanded through the use of bridges to form
AppleTalk internetworks.  AppleTalk bridges  are used to join two or
more AppleTalk networks.  In order for a bridge to identify each
AppleTalk  network,  a unique  AppleTalk  network  number is assigned
to each network joined by the bridge.
```

Furthermore, the AppleTalk Networking Architecture defines the concept of a zone. A zone is a subset of your AppleTalk Networks. You are free to assign a network to any one zone. Networks are usually assigned to a zone consisting of other networks that require access to the same shared devices. For example, a department may consist of several AppleTalk networks with only one Laserwriter for the entire department. In this case, you might want to group all the AppleTalk networks in the department into one zone.

* Type RETURN to continue:

9. Press RETURN.

The installation procedure displays the following warning:

```
                    ** WARNING **
If you install more than one bridge on the same
network, you must be sure to provide the same
network number and zone name for that network to both
bridges. For example, if you have installed an
AppleTalk Bridge on the Ethernet you must supply the
same network number and zone name to the
AppleTalk Ethernet network number and zone name
prompts below.

* Type RETURN to continue:
```

10. Press RETURN.

The installation procedure displays the following information:

```
AppleTalk for VMS Background Information
----------------------------------------

AppleTalk for VMS installs a Virtual AppleTalk Network on your
VAX. This network behaves like any other real AppleTalk network and
as such requires you to assign it a unique AppleTalk network number.
The AppleTalk Ethernet Network also requires you to assign it a
unique AppleTalk network number. In order to join the Virtual
AppleTalk Network and the AppleTalk Ethernet Network, AppleTalk
for VMS installs an AppleTalk Bridge process.

The AppleTalk for VMS network and zone name assignments are kept in
a port descriptor file, PORTDESC.TEXT, in the APPLETALK
directory. This file is created based upon answers provided by
you. The format of the file is as follows:
```

```
* Type RETURN to continue:
```

11. Press RETURN.

The installation procedure displays this information:

```
INIT:ATKAELAP
INIT:ATKVLAP
OPEN:PortName:ATKAELAP:ZoneName:NetworkNumber
OPEN:PortName:ATKVLAP:ZoneName:NetworkNumber

where "OPEN" & "INIT " indicate the type of line (a port must be
inited before it can be opened), "PortName" is unique port name
string, "ATKVLAP" & "ATKAELAP" are the port handler file names,
"ZoneName" is the name of the zone you  wish to assign to the
network, and "NetworkNumber" is  the network number in
hexadecimal (base 16).

The file is maintained as a standard VMS text file and can be
changed at any time using a text editor.  However, you should
make every effort to properly create the port descriptor file at
this time.

* Do you understand [Y]?
```

12. Enter YES.

The installation procedure displays the following sentence and series of questions:

```
The next set of questions will ask for the various pieces of the
port descriptor file and create the file based on your answers.

* AppleTalk Ethernet Port Name [AELAP PORT]:
* AppleTalk Ethernet Zone Name [VAX ZONE]:
* AppleTalk Ethernet Network Number in hexadecimal [0001]:
* Virtual AppleTalk Port Name [VLAP PORT]:
* Virtual AppleTalk Zone Name [VAX ZONE]:
* Virtual AppleTalk Network Number in hexadecimal [8000]:
```

13. Enter a response to each of the preceding questions, using the information you prepared before starting the procedure.

The installation procedure displays this text:

```
        **      Create AppleTalk's account ...

%VMSINSTAL-I-ACCOUNT, This installation creates an account named AppleTalk.

* Password for the AppleTalk account:
```

14. Enter the password for the APPLETALK account.

The installation procedure displays this warning:

```
                        ** Warning **
                UIC of AppleTalk account should not be changed if
                Appletalk account was created before.

* UIC to assign to the account (e.g. [1,100]):
```

15. Enter the UIC of the APPLETALK account.

The installation procedure displays lines that look like these:

```
%UAF-I-ADDMSG, user record successfully added
%UAF-I-RDBADDMSGU, identifier APPLETALK value: [000001,000100] added to
RIGHTSLI
ST.DAT
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%VMSINSTAL-I-SYSDISK, This product creates directory ATK$ROOT:[APPLETALK].

        **      Mark files to be moved (time consuming) ...


                        ******************
                        ** PLEASE NOTE **
                        ******************
```

```
If you want the AppleTalk for VMS to be started automatically when
your system is rebooted, you must edit the site-specific system
startup command file SYS$MANAGER:SYSTARTUP.COM and add the
following:

                    @SYS$MANAGER:STARTATALK

* Do you understand [YES]?
```

16. Enter YES.

The installation procedure displays lines that look like these:

```
**      Create AppleTalk.Ins_Log ...


      AppleTalk for VMS V02-00 installation completed successfully.

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target
directories...
Job LOGIN (queue SYS$BATCH, entry 1) started on SYS$BATCH

      Installation of ATKVMS V2.0 completed at 14:50


      VMSINSTAL procedure done at 14:51
```

If the installation procedure completes as shown, you have successfully installed version 2.0 of the AppleTalk for VMS software. For information about starting the software, see Chapter 4.

Chapter 3

AppleTalk for VMS Updating

Overview

This chapter describes how to update the AppleTalk for VMS software from an earlier version to version 2.0. For information on installing AppleTalk for VMS for the first time, see Chapter 2.

Updating AppleTalk for VMS consists of two main steps: preparing to update, and running the update procedure. The rest of this chapter describes these two main steps.

Important

The AppleTalk for VMS update should be performed only by the system manager. The system manager should be familiar with VMS software installation procedures, hidden devices (rooted directories), directory and file protection, Digital Command Language (DCL), and a text editor such as Digital's EDT software.

Preparing to update

Before updating AppleTalk for VMS, you should prepare your system by following the steps listed in this section. After completing these steps, you will be ready to run the AppleTalk for VMS update procedure.

To prepare your system for the update, follow these steps:

1. **Log in to the SYSTEM account of your VAX computer.**
2. **Make sure the VAX computer is running the following required software:**

   VMS operating system, version 4.6 or later

   VAX C run-time library, version 2.1 or later

Important

Make sure you have installed all software patches for the version of the VMS operating system you are running. Otherwise, AppleTalk for VMS will not operate properly. Installing all patches is especially important if you are running version 5.0 of VMS. For copies of VMS software patches, contact Digital Equipment Corporation.

---

3. **Make sure the SYSTEM account meets the following requirements:**

   The user name is SYSTEM.

   The user identification code (UIC) is [1,4].

   The account has access to all system privileges.

   All system privileges are enabled.

4. **Make sure you have enough free blocks of storage on your system disk.**

   AppleTalk for VMS requires 1500 blocks temporarily during installation; it requires 200 blocks permanently on the system disk.

5. **Back up your system disk.**

6. **Enter the following DCL command to find the name of the rooted directory currently storing the AppleTalk for VMS software:**

   ```
   $ SHOW LOGICAL ATK$ROOT
   ```

7. **Decide whether you want version 2.0 of the AppleTalk for VMS software to reside in the rooted directory now defined as ATK$ROOT. If not, decide the name of the new directory that will store the software.**

   Note that a rooted directory name must contain a trailing period (.)—for example, `DUA0:[SYS0.]` is a valid name for a rooted directory.

8. **Invoke the VMS utility AUTHORIZE to find the user identification code (UIC) of the APPLETALK account.**

   User identification codes and passwords are stored in the user authorization file (UAF). During the update procedure, you will be prompted for the password and UIC of the `APPLETALK` account. You may enter a new password to replace the one now in the UAF. Do not enter a new UIC; use the UIC shown in the user authorization file.

9. **Make sure that the port descriptor file is accurate.**

   The port descriptor file (`PORTDESC.TEXT`) is stored in the directory `ATK$ROOT:[APPLETALK]`. Edit this file to make sure it is still accurate. If it is not, make the necessary changes.

❖ *Note:* For information regarding parameters and descriptions of the different port handlers supplied with AppleTalk for VMS, refer to the *AppleTalk for VMS Protocol Support Library Reference Manual*, Appendix A.

When editing, check the following information for each port:

- the `Port Driver File Name`, the file name of a port driver shareable image (only the file name portion of the file specification is used)
- the `Port Information String`, a port-dependent information string that is optional for some port types.
- the `Port Name`, a user-specified string of ASCII characters that uniquely identifies a particular port.
- the `Zone Name`, the AppleTalk zone name being assigned to the port's network. The `Zone Name` is optional, and is valid only for ports connected to AppleTalk networks. However, when omitting the `Zone Name` but supplying a port information string, you must supply a colon to mark the omitted `Zone Name`.

Note that changes you make in the port descriptor file take effect only after you stop and restart the Bridge Process. The following brief description will help you edit the port descriptor file.

The port descriptor file consists of port descriptor lines, of which there are two types: `INIT` and `OPEN`. The `INIT` line initializes the specified port handler; each port handler must be initialized before it can be opened. The `OPEN` line opens the specified port handler.

The format of the `INIT` line is

```
INIT:Port Driver File Name:Port Information String
```

The format of the `OPEN` line is

```
OPEN:Port Name:Port Driver File Name:Zone Name:Port
Information String
```

When editing the port descriptor file, carefully consider the network and port assignments. For the virtual AppleTalk network to be accessible to other AppleTalk networks, the port descriptor file must have at least four port descriptor lines. For example, on a VAX using Ethernet to connect to other AppleTalk bridges, the port descriptor file must contain at least the following port descriptor lines:

```
INIT:ATKVLAP
INIT:ATKAELAP
OPEN:VLAP Port:ATKVLAP:Zone Name:Network Number
OPEN:AELAP Port:ATKAELAP
```

The `VLAP` port is used to link to the virtual AppleTalk network inside the VAX computer. `Zone Name` is the name of the zone you wish to assign to the virtual AppleTalk network, and `Network Number` is the hexadecimal number that you wish to assign to the virtual AppleTalk network. The network number must be unique in your AppleTalk internetwork.

The `AELAP` port enables the Bridge Process to communicate with other bridges on the Ethernet.

10. **Check that any communication hardware to be used with AppleTalk for VMS is working.**

11. **Make sure the batch queue `SYS$BATCH` is operating and not stopped.**

    ```
    $ SHOW QUEUE SYS$BATCH
    ```

    If this queue is stopped or not operating, AppleTalk for VMS will not start.

12. **Stop any currently active AppleTalk for VMS processes.**

    The installation procedure requires that all currently active AppleTalk for VMS processes be stopped. Among these processes are the Bridge Process and any layered products that use AppleTalk for VMS.

13. **Set the default directory to `SYS$UPDATE`.**

    ```
    $ SET DEFAULT SYS$UPDATE
    ```

14. **Load the AppleTalk for VMS version 2.0 distribution tape on a tape drive, but do not mount the tape.**

Running the update procedure

After preparing your system as described in the previous section, you run the AppleTalk for VMS update procedure. This section shows how to run the procedure. The update procedure is started by the VMSINSTAL command procedure (VMSINSTAL.COM), part of the VMS operating system. For general information about using VMSINSTAL or about installing VMS layered products, see the VAX/VMS documentation set, available from Digital Equipment Corporation.

For illustration, these instructions assume that your kit was distributed on magnetic tape and that you will use device MSA0:. If you are using a different distribution medium or a different device, substitute the name of the medium or device you are using.

❖ *Note:* All installations of AppleTalk for VMS software are recorded in the log file ATK$ROOT:[APPLETALK]APPLETALK.INS_LOG. You can examine this file to see a history of AppleTalk for VMS installations.

To run the AppleTalk for VMS update procedure, perform the following steps:

1. **Start the update command procedure by entering the following DCL commands at the VMS system prompt ($):**

```
$ SET DEFAULT SYS$UPDATE
$ @VMSINSTAL ATKVMS020 MSA0:
```

The update procedure displays several lines that look like these:

```
        VAX/VMS Software Product Installation Procedure V5.0


It is 25-AUG-1988 at 18:23.
Enter a question mark (?) at any time for help.

%VMSINSTAL-W-DECNET, Your DECnet network is up and running.
%VMSINSTAL-W-ACTIVE, The following processes are still active:
        SQA
        _TXA3:
        WOLFE
* Do you want to continue anyway [NO]? yes
```

2. **Enter YES.** (Except where noted in these instructions, entering NO causes the update procedure to terminate, returning you to the VMS system prompt.)

The update procedure displays this question:

```
* Are you satisfied with the backup of your system disk [YES]?
```

3. **Enter YES.**

The update procedure displays a question like this one:

```
Please mount the first volume of the set on  MSA0:.
* Are you ready?
```

4. **Enter YES after you have mounted the volume.**

The update procedure displays several lines that look like these:

```
%MOUNT-I-MOUNTED, ATKVMS mounted on _MSA0:

The following products will be processed:

  ATKVMS V2.0


        Beginning installation of ATKVMS V2.0 at 18:26

%VMSINSTAL-I-RESTORE, Restoring product saveset A...

        AppleTalk for VMS, Version V02-00
        A product of Apple Computer Inc.

        This installation installs the following applications:

        AppleTalk for VMS Library              Version: V02-01A
        AppleTalk Bridge Process               Version: V02-02A
        Virtual AppleTalk Network Driver       Version: V02-01A
        AppleTalk Bridge Control Program       Version: V02-01A

* Please enter your name:
```

5. **Enter your name.**

The update procedure displays this question:

```
* Do you want to purge files replaced by this installation [YES]?
```

6. **Enter YES.** (If you enter NO, the update procedure continues and does not purge the files.)

The update procedure displays this text:

```
The installation of AppleTalk for VMS requires that any currently
active VMS AppleTalk applications be stopped. These applications
include the AppleTalk for VMS bridge process, and any layered products
which use the AppleTalk for VMS system. Answering NO to the
following prompt will abort the installation so you can stop the
AppleTalk applications. Answering YES will assume that all AppleTalk
applications have already been stopped.

* OK to continue with the AppleTalk for VMS installation. [Y]?
```

7. **Enter YES.**

The update procedure displays this question:

```
* Do you want to use existing ATK$ROOT definition [Y]?
```

8. **Enter YES to use the current definition.** (Entering NO allows you to change the directory name.)

The update procedure displays this information:

```
AppleTalk Networking Background Information
-------------------------------------------

AppleTalk networks  can be  expanded through the use of bridges to form
AppleTalk internetworks.  AppleTalk bridges  are used to join two or
more AppleTalk networks.  In order for a bridge to identify each
AppleTalk  network,  a unique  AppleTalk  network  number is assigned
to each network joined by the bridge.
```

```
Furthermore, the AppleTalk Networking Architecture defines the
concept of a zone. A zone is a subset of your AppleTalk Networks. You
are free to assign a network to any one zone. Networks are usually
assigned to a zone consisting of other networks that require
access to the same shared devices. For example, a department may
consist of several AppleTalk networks with only one Laserwriter
for the entire department. In this case, you might want to group
all the AppleTalk networks in the department into one zone.

* Type RETURN to continue:
```

9. Press RETURN.

The update procedure displays this warning:

```
                    ** WARNING **
If you install more than one bridge on the same
network, you must be sure to provide the same
network number and zone name for that network to both
bridges. For example, if you have installed an
AppleTalk Bridge on the Ethernet you must supply the
same network number and zone name to the
AppleTalk Ethernet network number and zone name
prompts below.

* Type RETURN to continue:
```

10. Press RETURN.

The update procedure displays this information:

```
AppleTalk for VMS Background Information
----------------------------------------

AppleTalk for VMS installs a Virtual AppleTalk Network on your
VAX. This network behaves like any other real AppleTalk network and
as such requires you to assign it a unique AppleTalk network number.
The AppleTalk Ethernet Network also requires you to assign it a
unique AppleTalk network number. In order to join the Virtual
AppleTalk Network and the AppleTalk Ethernet Network, AppleTalk
for VMS installs an AppleTalk Bridge process.

The AppleTalk for VMS network and zone name assignments are kept in
a port descriptor file, PORTDESC.TEXT, in the APPLETALK
directory. This file is created based upon answers provided by
you. The format of the file is as follows:
```

```
* Type RETURN to continue:
```

11. Press RETURN.

The update procedure displays this information:

```
INIT:ATKAELAP
INIT:ATKVLAP
OPEN:PortName:ATKAELAP:ZoneName:NetworkNumber
OPEN:PortName:ATKVLAP:ZoneName:NetworkNumber
```

```
where "OPEN" & "INIT " indicate the type of line (a port must be
inited before it can be opened), "PortName" is unique port name
string, "ATKVLAP" & "ATKAELAP" are the port handler file names,
"ZoneName" is the name of the zone you  wish to assign to the
network, and "NetworkNumber" is  the network number in
hexadecimal (base 16).

The file is maintained as a standard VMS text file and can be
changed at any time using a text editor.  However, you should
make every effort to properly create the port descriptor file at
this time.

* Do you understand [Y]?
```

12. Enter YES.

The update procedure displays lines that look like these:

```
The following port descriptor file already exists.  You may choose
to use the existing port descriptor file or create a new one.

                        ** WARNING **
                The existing port descriptor  file must follow  the
                format shown above.  If your existing file does NOT
                conform to that format,  you must create a new one.
                Pay  special  attention to the  port  handler  file
                names.

INIT:ATKAELAP
OPEN:ETHERTALK:ATKAELAP:ETHERKNOTT:6979
INIT:ATKVLAP
OPEN:VIRTUALTALK:ATKVLAP:NETS-R-US:6B44

* Do you want to use this port descriptor file [YES]?
```

13. Enter YES.

The update procedure displays this text:

```
        **      Create AppleTalk's account ...

%VMSINSTAL-I-ACCOUNT, This installation creates an account named AppleTalk.

* Password for the AppleTalk account:
```

14. Enter the password for the APPLETALK account.

The update procedure displays this warning:

```
                        ** Warning **
                UIC of appletalk account should not be changed if
                appletalk account was created before.

* UIC to assign to the account (e.g. [1,100]):
```

15. Enter the UIC of the APPLETALK account.

The update procedure displays lines that look like these:

```
%UAF-I-ADDMSG, user record successfully added
%UAF-I-RDBADDMSGU, identifier APPLETALK value: [000001,000100] added -
to RIGHTSLIST.DAT
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated
%UAF-I-MDFYMSG, user record(s) updated

        **      Set owner of existing AppleTalk files to [1,100] ...

        **      Mark files to be moved (time consuming) ...

                        ******************
                        ** PLEASE NOTE **
                        ******************
```

```
        If you want the AppleTalk for VMS to be started automatically
        when your system is rebooted, you must edit the site-specific
        system startup command file SYS$MANAGER:SYSTARTUP.COM and add
        the following:

                @SYS$MANAGER:STARTATALK

* Do you understand [YES]?
```

16. Enter YES.

The update procedure displays lines that look like these:

```
        **      Update AppleTalk.Ins_Log ...


        AppleTalk for VMS V02-00 installation completed successfully.

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...
Job LOGIN (queue SYS$BATCH, entry 1025) started on SYS$BATCH

        Installation of ATKVMS V2.0 completed at 18:32


        VMSINSTAL procedure done at 18:33
```

If the update procedure completes as shown, you have successfully run the update procedure. You now continue the update of AppleTalk for VMS by reloading the Virtual AppleTalk Network Driver.

17. Reload the Virtual AppleTalk Network Driver by rebooting the VMS operating system.

For more information about starting the AppleTalk for VMS software, see Chapter 4.

Chapter 4

AppleTalk for VMS Operation

Overview

This chapter describes how to start up and shut down AppleTalk for VMS processes. The chapter also describes how to create multiple virtual networks and how to restart the Bridge Process.

The startup procedure

A series of command files must be executed at the time of VAX/VMS system startup to operate AppleTalk for VMS.

To establish this startup procedure, the VMS file `SYSTARTUP` must be edited by adding a line that calls the AppleTalk startup command file, as follows:

```
$ @SYS$MANAGER:STARTATALK.COM
```

❖ *Note:* If DECnet™ networking sofware is to be run on the same Ethernet controller as AppleTalk for VMS, the AppleTalk startup line must be entered *after* the DECnet startup line, so that DECnet is started before the AppleTalk for VMS network. If you wish to start DECnet in a batch job, create a command file containing the following two lines, in the order listed:

```
$ @STARTNET.COM          ! start DECnet
$ @STARTATALK.COM        ! start AppleTalk
```

You may then submit this command file for batch processing, instead of submitting `STARTNET.COM` alone.

The `STARTATALK.COM` command file initiates the following functions:

- assigns the logical `ATK$ROOT` to the directory defined at installation
- installs the AppleTalk for VMS run-time library as a shareable image (using the VMS `INSTALL` utility)
- installs the port driver images (using the VMS `INSTALL` utility)
- loads the Virtual AppleTalk Driver (using the VMS `SYSGEN` utility)
- submits `ATK$ROOT:[APPLETALK]LOGIN.COM` as a batch job. This batch job executes `ATK$ROOT:[APPLETALK]STARTBRIDGE.COM`,which starts the AppleTalk Bridge Process

Multiple virtual networks

Use the VMS SYSGEN utility to load additional copies of the Virtual AppleTalk Driver (the VN: driver). Any additional VN: driver must be connected to a separate controller, and each controller must be identified as A, B, C, and so on in the STARTATALK.COM file, as follows:

```
$ RUN SYS$SYSTEM:SYSGEN
      LOAD VNDRIVER
      CONNECT VNx:/NOADAP
```

where x is the controller identifier (A, B, or C).

A separate VLAP port must be opened so that the Bridge Process will recognize the new virtual network driver. This is done by creating an OPEN line for the VLAP port in the port descriptor file (see "Preparing for installation" in Chapter 1). The port information string of the OPEN line must specify the following:

```
network number,controller ID
```

with `network number` supplied in hexadecimal notation.

Restarting the Bridge Process

If the Bridge Process is brought down by a system fault or user error, it is not necessary to rerun the entire system startup file, STARTATALK.COM, to restart the Bridge Process. Instead, the command to submit the startup batch job may be given from the DCL prompt, as follows:

```
SUBMIT/NOLOG/NOPRINTER/USER=APPLETALK atk$root:[appletalk]login/param=DETACH
```

This command should be submitted from the APPLETALK account only.

Shutdown

You can shut down all AppleTalk for VMS processes or just a single process.

Shutting down all processes

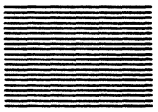
To stop all AppleTalk for VMS processes, you perform a normal VAX/VMS system shutdown—that is, you shut down the entire VAX/VMS system. Note that you need not prepare AppleTalk for VMS before shutting down the VAX/VMS system. Such a shutdown neither corrupts AppleTalk for VMS files nor damages the AppleTalk for VMS software.

Shutting down a single process

To stop a single AppleTalk for VMS process, you enter the DCL command `STOP/ID`, which has the following syntax:

```
STOP/ID = pid
```

Here, `pid` is the process identification number of the process to be stopped—for example, `STOP/ID = 00002A`. To find out the identification number of a currently running VMS process, enter the DCL command `SHOW SYSTEM`. The AppleTalk for VMS processes are `ATK/VMS Bridge` and any additional AppleTalk layered products.



Part 2

AppleTalk Data Stream Protocol

Chapter 5

Using ADSP

Overview

The major new feature of AppleTalk for VMS, version 2.0, is the addition of the AppleTalk Data Stream Protocol (ADSP) to the protocol support library. This chapter and those that follow describe the VMS client interface to ADSP. These chapters provide details about the VMS implementation of ADSP only when needed to explain the ADSP interface. For a more detailed specification of the AppleTalk Data Stream Protocol, consult the *AppleTalk Data Stream Protocol Preliminary Note,* available from Apple Computer, Inc.

General features of ADSP

ADSP is a symmetric, connection-oriented protocol that makes possible the establishment and maintenance of full-duplex streams of data bytes between two sockets in an AppleTalk internet. ADSP ensures reliable data flow by delivering data bytes free of duplicates, in the same order as they are sent. In addition, ADSP includes a flow-control mechanism that uses information supplied by the intended destination socket.

Using ADSP, the client can open a connection between a local socket and a remote socket, send data to and receive data from the remote socket, and close the connection. The client can either send a continuous stream of data or logically break the data into client-intelligible messages. Additionally, ADSP provides an attention-message mechanism that the client can use for its own internal control. A forward-reset mechanism allows the client to abort the delivery of an outstanding stream of bytes to the remote client.

The ADSP protocol's connection-opening features include the ability to establish a connection-listening socket that diverts connection establishment to a third party. Also included is the ability to limit the sockets with which a connection may be established. The client may also bypass the internal ADSP connection-opening mechanism by substituting an external, client-supplied mechanism.

The VMS implementation of ADSP

The VMS implementation of ADSP is a set of procedure calls that provide access to all of the functionality of ADSP. Specifically, procedure calls are implemented that provide for

- opening and closing a connection
- sending/receiving data bytes
- breaking the data stream into logical messages
- sending/receiving attention messages
- forward-resetting the connection
- implementing connection listeners

- receiving/accepting/denying connection requests
- obtaining connection status

Several additional calls are provided for tuning ADSP, observing ADSP's internal significant events, and managing buffers.

Opening and closing a connection

Connection opening is the process of creating an association between two sockets in an internet to allow the reliable exchange of data streams. Connection-end information and its associated socket constitute a *connection end.* Connection-end information includes the connection identifier (CID) and the values of connection-state variables.

Normally, the opening process requires a connection-establishment dialog between the two sockets to synchronize each connection end's state variables. State variables include the size of each end's receive queues and the initial sequence numbers of the data streams. The VMS implementation provides three procedure calls to establish a connection: `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, and `ATK$DSP_ESTABLISH`. Each procedure implements a different method of establishing a connection, based on the connection initiator and the availability of connection-end synchronizing information prior to establishing the connection.

`ATK$DSP_OPEN` is used to establish a connection with a specified remote socket. It actively seeks to establish a connection by initiating the opening dialog between the local and remote ADSP modules. No prior knowledge of any connection-end state variables is required to use this procedure.

`ATK$DSP_LISTEN` is the functional opposite of `ATK$DSP_OPEN`. Rather than actively seeking to open the connection, `ATK$DSP_LISTEN` creates a connection end on a socket and waits for some other remote socket to initiate the opening dialog. `ATK$DSP_LISTEN` also does not require any prior knowledge of connection-end state variables.

Both `ATK$DSP_OPEN` and `ATK$DSP_LISTEN` allow the client to select the remote sockets with which the connection may be established. The connection initiator may specify a list of address filters against which a designated socket will be checked prior to connection establishment. Connection-denial messages are sent to those sockets that are not acceptable.

`ATK$DSP_ESTABLISH` forces a connection to become immediately established. Within ADSP, no connection-opening dialog is initiated or received, and the client must provide all the information necessary to place the local connection end in a synchronized state. The client is also responsible for synchronizing the remote connection end. This form of connection establishment bypasses the ADSP connection-opening mechanism and allows the client to use a connection-opening dialog external to ADSP.

Prior to establishing a connection using `ATK$DSP_ESTABLISH`, an ADSP client must first create a connection end upon which to establish a connection. The procedure call `ATK$DSP_CREATE_CON_END` creates a connection end and returns the information necessary for the client to establish the connection at the remote end.

❖ *Note:* Even when a connection is explicitly denied by the remote ADSP module, the ATK$DSP_OPEN procedure continues to try to establish the connection. The number of retries is specified by the opening parameter ATK$K_OPENMAXRETRY.

When establishing a connection, a client may also specify the address of a procedure that will be called whenever a significant event occurs on the connection. Once the connection is actually established, this connection-event handler will be called under the following circumstances:

- the connection receives a connection-close advice from the remote end
- the connection is torn down because the remote end is unreachable
- a forward-reset message is received
- an attention message is received

To break down a connection, the procedure ATK$DSP_CLOSE is used. ATK$DSP_CLOSE can be directed to break the connection immediately, or to wait until the send queues have emptied.

Sending and receiving data bytes

Once the connection is established, the flow of data can be started. Four procedure calls are implemented for reading and writing data to the connection: ATKDSP_READ, ATKDSP_WRITE, ATK$DSP_GET_BYTE, and ATK$DSP_PUT_BYTE.

ATK$DSP_READ and ATK$DSP_WRITE are used to transmit and receive contiguous buffers of data bytes. ATK$DSP_PUT_BYTE and ATK$DSP_GET_BYTE read and write a single byte at a time. The Put/Get Byte interface is nonblocking, while the Read/Write interface allows the client to post a read request and be notified when a buffer of data has been received.

Each pair of calls implies a different buffer allocation scheme. With the buffer read and write routines, the client must allocate and deallocate buffers. With the byte routines, ADSP will allocate and deallocate buffers as necessary on the client's behalf. These two methods are further described in the discussion of buffer management internals later on in the section "Buffer Management."

Breaking the data stream into logical messages

Depending on the application, the ADSP client may need to mark message boundaries in the data stream. When writing data to a connection using ATK$DSP_WRITE, the client may pass a flag to ADSP indicating that the last byte in the buffer is the last byte of a client-intelligible message. ADSP will then inform the remote connection end about the end of message by inserting a logical end of message marker into the data stream immediately following the last byte of the message. The remote client will be notified of the end of message when it extracts the logical end of message marker from the connection.

When using ATK$DSP_PUT_BYTE to transmit data, a client cannot indicate an end-of-message. In this case the client must make a call to ATK$DSP_SEND_EOM to indicate that the last byte inserted into the connection was the last byte of a logical message. ATK$DSP_SEND_EOM also works with ATK$DSP_WRITE.

Sending and receiving attention messages

A client may need to send a message to the remote client outside of the normal data stream. The ADSP protocol provides for such out-of-band transmissions with its attention-message mechanism. Two procedure calls are implemented that allow the client to send and receive attention messages: ATK$DSP_SEND_ATTENTION and ATK$DSP_RECV_ATTENTION.

The connection-event handler will notify the client asynchronously when an attention message has been received. In this way, the client need only post an attention-message read when notified that one is available.

Forward-resetting the connection

During normal processing, the client may detect that the information just sent to the remote client is no longer useful. To respond to this situation, ADSP provides a mechanism for resetting the connection, purging the receiving connection end of any data bytes not yet delivered, and informing the remote client that the connection has been reset. This forward-reset mechanism is implemented for VMS with the procedure ATK$DSP_FORWARD_RESET.

Implementing connection listeners

A *connection listener* is a client of ADSP. This client uses a connection-listening socket on which to implement a listener function, which will divert incoming open connection requests to other sockets, as shown in Figure 5-1 and described below.

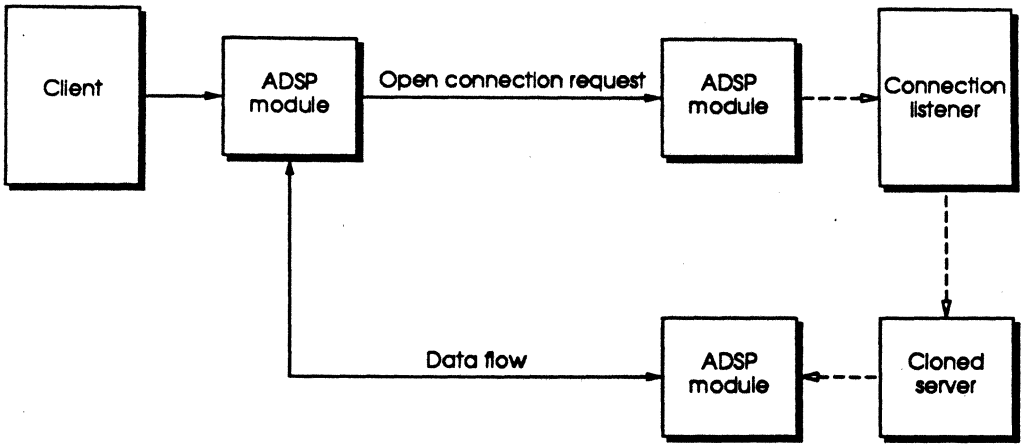


Figure 5–1
A connection listener implementation

A typical use of connection listeners is in the implementation of network-based servers (file, print, block, etc.). Such an implementation may require a separate copy of the server to be spawned for each active client–server session. A single process may field all open connection requests, clone a new copy of the requested server for each, and then divert each request to its new server. The server completes the connection with the remote client, and the data transfer may then begin. In this process, the connection listener uses the open connection request information to establish the actual connection on another socket, which may be on another node or network altogether.

Establishing connection listeners

The VMS implementation of ADSP provides two procedure calls for creating and deleting a connection-listening socket. These are `ATK$DSP_CREATE_LIS_SOC` and `ATK$DSP_DELETE_LIS_SOC`. Once a connection listener is established for a socket, future open connection requests received by that socket will be passed on intact to the client (normally, the ADSP implementation on the socket would handle open connection requests and responses internally, but the existence of the connection listener causes the request to pass to the client).

The ADSP protocol enables the connection listener to divert the open connection request to a different socket which can complete the connection with the requesting client.

When a connection listener is present, the initiator of an open connection request may be unable to ascertain in advance whether the target socket is the socket where the connection is finally established. Depending on the initiating client's application, this condition may not be acceptable. To solve this problem, the connection initiator may specify a list of address filters. When the open connection request is received by the connection initiator, the source socket is checked against the filter list. If the source socket is acceptable, the connection is established. Otherwise, a connection denial is transmitted.

Receiving, accepting, and denying connection requests

Three procedures are provided to connection listeners for receiving, accepting, and denying open connection requests. ATK$DSP_GET_CON_REQ is used to receive an open connection request from a listening socket. The information given to the client contains all the pertinent fields of the open connection request. ATK$DSP_ACCEPT is the procedure to use when the connection will be established on another socket. The client must supply the corresponding open connection request fields to this routine. ATK$DSP_DENY_CON_REQ will turn down an unacceptable connection request.

- *Note:* A connection request may be denied from either the connection listening socket or the newly created socket.

The VMS implementation does not restrict a connection listener from establishing a connection on the connection-listening socket. However, it does require that connection-listening operations be discontinued on the socket before the connection is established.

Connection listeners bypass the mechanism by which ADSP detects duplicate connection requests. Therefore, if the same request is sent more than once, the connection listener must itself detect and ignore the duplicate requests. Duplicate connection requests occur when the remote end sends a request but receives no response before a timeout interval expires. The remote end therefore retransmits the request, causing a duplicate connection request.

One method by which connection listeners can detect and ignore duplicate requests is by keeping a list of active connections and checking each request against this list. That is, upon receiving each request, the connection listener examines the remote connection identifier and remote internet address of the request. If the list of active connections already contains an entry matching both these values, the request is ignored as a duplicate. Entries are removed from the list of active conections when the connection represented is no longer active.

Obtaining connection status

The current status of a connection can be obtained by a call to the procedure ATK$DSP_GET_STATUS. The connection end's current state and sequence numbers are returned. A client may use this information to determine (among other things) whether the send queue has been emptied before closing a connection. The status information returned is described in Appendix B.

Tuning ADSP

Several implementation-specific operating parameters affect the performance of ADSP. Some of these parameters affect the local ADSP module's relationship with the remote ADSP module, while others affect the local ADSP module's internal management. Six procedure calls are provided to manipulate these parameters.

The first procedure, `ATK$DSP_GET_DEFAULTS`, is used to obtain a copy of the current default operating parameters. Two different lists may be obtained depending on the client's needs: the baseline list or the current operating defaults list.

The baseline list is the set of baseline operating parameters. This list can never be changed, and represents the factory-default settings for ADSP. A client can always refer to this list to get back to baseline operation. The client may also obtain a copy of this list from ADSP, modify selected fields, and use the new list to set up the current operating defaults list, or the operating parameters of a specific connection.

The second procedure, `ATK$DSP_SET_DEFAULTS`, is used to set the current operating defaults list. This list represents the operating parameters to be used for a connection if not explicitly supplied in the connection-establishment call. A client may set the operating parameters in this list, after which all future connections established will use this list.

The easiest way to use `ATK$DSP_SET_DEFAULTS` is to first obtain a full list from `ATK$DSP_GET_DEFAULTS`, modify the appropriate fields, and then set the current operating defaults.

The next four procedures are the actual connection-establishment procedures. `ATK$DSP_OPEN`, `ATK$DSP_LISTEN`, `ATK$DSP_ACCEPT`, and `ATK$DSP_ESTABLISH` all allow the client to specify the operating parameters that will be used to maintain the connection.

If the list is not specified, or parts of the list are missing, the current operating defaults list will be used. If the current operating defaults list is not specified, the baseline list is used. A separate copy of the operating parameters list exists for each active connection.

The routines which manipulate the tuning parameters use a special list to pass the necessary parameters back and forth. The format of the list and how it can be manipulated are described later in the section "Using Tagged Lists." Specific tuning parameters are discussed in Appendix A.

Breaking a connection

The breakdown of a connection begins with one of three events:

- The local client issues an `ATK$DSP_CLOSE` request.
- The local connection end receives a close advice message.
- The remote connection end becomes unreachable.

❖ *Note:* If the breakdown begins when the remote connection end becomes unreachable, the process of breaking the connection is also known as ***tearing down*** the connection.

After any of the events that begin a breakdown, ADSP changes the state of the connection, reflecting the kind of event that began the breakdown. (The state of the connection is maintained internally by ADSP.) ADSP then attempts to notify the client that the event has occurred. This attempt succeeds only if the client has an active request posted to ADSP, or has an active connection-event handler. If this attempt to notify the client fails, the client is notified the next time it issues a request to ADSP.

During a breakdown, a client can perform only some of the operations that it normally can perform on a connection. Exactly which operations the client can perform depends on the breakdown mechanism—that is, on which kind of event begins the breakdown. If the breakdown begins because the client issued an `ATK$DSP_CLOSE` request whose wait parameter equals zero, the connection end is immediately deleted; therefore, the client cannot perform any operations on the connection. If, however, the wait parameter of the request does not equal zero, the client can perform only the operation `ATK$DSP_GET_STATUS`.

If the breakdown begins because the local connection end received a close advice message, or because the remote connection end became unreachable, the client can perform `ATK$DSP_GET_STATUS` and the following receive operations: `ATK$DSP_READ`, `ATK$DSP_GET_BYTE`, and `ATK$DSP_RECV_ATTENTION`.

The breakdown mechanism also governs the deletion of the send queues, receive queues, and local connection end. If the breakdown begins because the local client received a close advice message or because the remote connection end became unreachable, the send queues are deleted immediately. The receive queues, however, remain intact until the local client either *(a)* issues an `ATK$DSP_CLOSE` request, or *(b)* reads all data in the receive queue, and is notified of the breakdown.

If the breakdown begins because the local client issued an `ATK$DSP_CLOSE` request, the deletion of the connection end, send queues, and receive queues is governed by the wait parameter of the request. If the wait parameter is nonzero, the send queues and receive queues are deleted only after the remote client acknowleges receipt of all sent data. The local conection end is then deleted after the send queues are empty. If, on the other hand, the wait parameter equals zero, the send queues and receive queues are deleted immediately, including any data not yet delivered to the clients. The local connection end is deleted immediately after deletion of the queues.

Observing ADSP's internal significant events

ADSP for VMS provides two procedures for observing ADSP's internal significant events. `ATK$DSP_GET_COUNTERS` returns a list of ADSP counters. This list is a series of counters that describe the significant events that ADSP has incurred. `ATK$DSP_CLEAR_COUNTERS` is used to reset the counters. Appendix C describes each of the counters.

Synchronizing client operations with ADSP

The VMS implementation of ADSP allows the client to be notified of the completion of requested operations either synchronously or asynchronously. Some of the ADSP procedure calls, by their nature, can only be completed synchronously. These procedures return control to the client only after the requested operation is completed. The synchronous procedures are

- ATK$DSP_CREATE_CON_END
- ATK$DSP_ESTABLISH
- ATK$DSP_WRITE
- ATK$DSP_PUT_BYTE
- ATK$DSP_GET_BYTE
- ATK$DSP_SEND_EOM
- ATK$DSP_FLUSH
- ATK$DSP_CREATE_LIS_SOC
- ATK$DSP_DELETE_LIS_SOC
- ATK$DSP_DENY_CON_REQUEST
- ATK$DSP_GET_BUF
- ATK$DSP_RET_BUF
- ATK$DSP_SET_DEFAULTS
- ATK$DSP_GET_DEFAULTS
- ATK$DSP_GET_STATUS
- ATK$DSP_GET_COUNTERS
- ATK$DSP_CLEAR_COUNTERS

These procedures manipulate local data structures and, in some cases, may initiate an I/O operation. In all cases, control is returned quickly to the client. The client is informed of the success of the requested operation through the function call return mechanism. An example is ATK$DSP_WRITE. A client making a call to ATK$DSP_WRITE is requesting ADSP to place a buffer of data bytes in ADSP's internal send queue. Once the data is in the send queue, ADSP may or may not transmit the data immediately to the remote end, depending on the state of the remote end's receive queue. Regardless of whether the data is transmitted, control is returned to the client immediately and the operation is considered complete. The client can proceed with the assurance that the data will be received by the remote client at some point in the future.

Because the balance of the ADSP procedures could conceivably take a long time to complete, the client may wish to perform other functions while the procedure is in operation. Procedures that allow the client to work in parallel with the procedure's operations are

- ATK$DSP_OPEN
- ATK$DSP_LISTEN

- ATK$DSP_ACCEPT
- ATK$DSP_CLOSE
- ATK$DSP_READ
- ATK$DSP_SEND_ATTENTION
- ATK$DSP_RECV_ATTENTION
- ATK$DSP_FORWARD_RESET
- ATK$DSP_GET_CON_REQ

The client may choose to wait for these procedures to complete before continuing, or to operate in parallel with the procedure's operation. If the client chooses to operate in parallel, a mechanism is provided that notifies the client when the operation has completed, and whether it has succeeded.

Four parameters can be provided to each of the procedures described above, which serve to synchronize the client's operation with the procedure:

- `CompStatus` is the address of a longword that will receive the completion status of a requested operation. Since asynchronous ADSP procedures return control to the client immediately after initiation, the `CompStatus` parameter is used to acquire their final completion status.
- `EventFlag` is the number of a VMS event flag. The specified event flag is cleared when the operation is initiated, and set when the operation is completed.
- `ASTHandler` is the address of a client procedure that will be called once the operation is complete. This procedure is executed at AST level and is passed the parameter `ASTParam` by value. The client may use `ASTHandler` to perform any client-specific cleanup.
- `ASTParam` is any client-specified value that will fit in a single longword. The value is passed to `ASTHandler` once the operation completes. This parameter may be used by the client with a general purpose `ASTHandler` procedure to distinguish which among many active operations has actually completed.

To use ADSP procedures in parallel, a client must specify either an `EventFlag`, an `ASTHandler`, or both. If neither are specified, the procedures will run to completion before returning control to the client.

Buffer management

When using `ATK$DSP_READ` and `ATK$DSP_WRITE`, the client is required to allocate buffers for transmission and to deallocate buffers that have been delivered to the client.

Two procedures are provided for client buffer management. `ATK$DSP_GET_BUF` is used to allocate a buffer from the memory pool, which the client uses for data transmission. `ATK$DSP_RET_BUF` is used to deallocate buffers obtained from `ATK$DSP_GET_BUF`, `ATK$DSP_READ`, or `ATK$DSP_RECV_ATTENTION`.

Buffer management internals

The VMS implementation of ADSP provides two interfaces for transmitting to and receiving data from a remote client in the normal data stream. The Put/Get Byte interface allows the transmission/reception of a single byte of data. The Read/Write interface allows buffers of multiple data bytes to be transmitted and received. The Put/Get Byte interface is nonblocking, while the Read/Write interface allows the client to post a read request and be notified when a buffer of data has been received.

When using the Put/Get Byte interface, all buffer management is performed by ADSP on the client's behalf. In actual implementation, ADSP maintains a linked list of buffers as the send/receive queues. The first time a client issues a Put Byte request, ADSP allocates a buffer from a memory pool which it shares with the client, and inserts the buffer at the end of the send queue. The size of the buffer is specified by the client as a tuning parameter. ADSP then copies the data byte to the buffer; succeeding Put Byte requests will continue to fill up the original buffer. Once the buffer is filled, a new one is allocated and placed at the end of the queue.

As bytes are removed from the send queue (by remote client acknowledgments), ADSP checks for empty buffers. Emptied buffers are returned to the memory pool.

When bytes are received from the remote client in DDP (Datagram Delivery Protocol) data packets, the packets are linked into the tail of the receive queue. Each time the client issues a Get Byte request, the next byte at the head of the queue is delivered. As data packets are emptied, they are returned to DDP. A Get Byte request to an empty queue returns an error code.

Since a DDP data packet has a capacity of 586 bytes, and the entire packet is linked into the receive queue, there is a potential for wasted space if the remote client delivers only a small number of bytes in each packet. ADSP allows the client to specify the maximum acceptable level of waste via a tuning parameter. This parameter sets the byte count level at which ADSP will copy data bytes from the next received packet into the previous packet, rather than just linking the new packet into the receive queue. The parameter is specified as a byte count threshold. Packets with fewer bytes than this threshold are copied, while packets exceeding the threshold are linked.

When using the Read/Write interface, the buffer management strategy perceived by the client is that of a common memory pool. A client wishing to transmit data first allocates a buffer from the memory pool, fills it with data, then requests ADSP to transmit the data. At this point, ADSP accepts the buffer into its send queue for transmission. The buffer is now part of the send queue and the client may no longer access it. Eventually, the data will be received by the remote client and ADSP will release the buffer back to the memory pool.

When the client issues a Read request, the next DDP packet at the head of the receive queue is unlinked and delivered to the client, along with a pointer to the start of the data in the packet. When finished with the packet, the client requests ADSP to return the buffer to the memory pool. If the receive queue is empty when the client issues a read request, the request is posted; it completes when ADSP receives the next DDP packet from the remote end.

Note that ADSP does not limit the amount of data that the local client inserts into the send queue. Instead, the local client must itself limit this amount, thereby conserving virtual memory.

Important

If the local client repeatedly inserts data into the send queue without waiting for the remote client to reduce the size of the queue, the local client can consume all the available virtual memory of the process. Such a lack of virtual memory greatly reduces the throughput of ADSP, possibly causing unpredictable results. However, a lack of process virtual memory does not compromise the reliable data transfer offered by ADSP.

One way the local client can limit its input to the send queue is to monitor the state variables of the send queue. In this method, the local client sets a maximum size for the send queue. If the state variables indicate that the send queue has reached this maximum size, the local client stops inserting data into the send queue. After the remote client reduces the queue to some specified size, the local client resumes inserting data into the send queue.

Using tagged lists

Tagged lists are used to pass a large number of parameters of different lengths and meanings through a procedure call interface. A *tagged list* is a sequence of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the entry field. The length of each list is defined as a constant.

ADSP uses tagged lists to pass default operating parameters, connection status, and ADSP internal counters back and forth to the client.

The number of parameters passed through a procedure call via a tagged list may be increased without affecting the caller of the procedure. From the caller's point of view, only the size of the list will have grown. If the caller has no use for the new parameters at the end of the list, they may simply be ignored. Provided the caller does not make any assumptions about the order of the parameters in the list—but uses tags for identification—the order of parameters can be changed by ADSP without affecting the client.

Chapter 6

ADSP Call Formats

This chapter provides call formats for the routines that follow.

```
ATK$DSP_ACCEPT
ATK$DSP_CLEAR_COUNTERS
ATK$DSP_CLOSE
ATK$DSP_CREATE_CON_END
ATK$DSP_CREATE_LIS_SOC
ATK$DSP_DELETE_LIS_SOC
ATK$DSP_DENY_CON_REQ
ATK$DSP_ESTABLISH
ATK$DSP_FLUSH
ATK$DSP_FORWARD_RESET
ATK$DSP_GET_BUF
ATK$DSP_GET_BYTE
ATK$DSP_GET_CON_REQ
ATK$DSP_GET_COUNTERS
ATK$DSP_GET_DEFAULTS
ATK$DSP_GET_STATUS
ATK$DSP_LISTEN
ATK$DSP_OPEN
ATK$DSP_PUT_BYTE
ATK$DSP_READ
ATK$DSP_RECV_ATTENTION
ATK$DSP_RET_BUF
ATK$DSP_SEND_ATTENTION
ATK$DSP_SEND_EOM
ATK$DSP_SET_DEFAULTS
ATK$DSP_WRITE
```

In addition to the call formats, this chapter provides a brief description of each routine and explains each of its arguments.

ATK$DSP_ACCEPT – Accept a connection with a remote end

ATK$DSP_ACCEPT accepts a connection with a remote socket based on parameters received by a connection listener.

Format

```
ATK$DSP_ACCEPT  Refnum,RemoteAddress,RemoteCID,PktNextRecvSeq
                ,PktRecvWdw,PktAttnRecvSeq,CompStatus
                [,Socket] [,ConEventHandler] [,EventFlag]
                [,ASTHandler] [,ASTParam] [,OPDefaults]
```

Arguments

Refnum

type: longword
access: write only
mechanism: by reference

The connection reference number. Refnum identifies the connection end for all future operations on the connection. The value of Refnum is available immediately upon return from this call, regardless of whether the call was issued synchronously.

RemoteAddress

type: unspecified
access: read only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

RemoteCID

type: word
access: read only
mechanism: by value

The 16-bit identifier of the connection end that is the target of the Open request. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket.

PktNextRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next data byte the remote connection end will accept into its receive queue. If it is omitted, an error is returned.

PktRecvWdw

type: word
access: read only
mechanism: by value

The size of remote connection end's receive queue. If it is omitted, an error is returned.

PktAttnRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to establish the connection. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes. If `CompStatus` is omitted, an error is returned.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the address of `Socket` is specified as zero or is not supplied, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of `Socket` is zero, ADSP will open a dynamic socket for the connection end. If the value of `Socket` is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for `Socket`. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. If the address of the `Socket` argument is supplied, and is nonzero, the connection socket number will be returned.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format `ConEventHandler SigEvent,Refnum`

Arguments `SigEvent`

type: longword
access: read only
mechanism: by value

The type of significant event that has occurred. `SigEvent` will have one of four values: `ATK__ATTMSGRCV`, which indicates the receipt of an attention message; `ATK__FORRESRCV`, which indicates the receipt of a forward reset; `ATK__CLOADVRCV`, which indicates the receipt of a close advice from the remote end; and `ATK__REMENDUNR`, which indicates the connection being torn down due to inability to reach the remote end.

Refnum

type:	longword
access:	read only
mechanism:	by value

The connection reference number identifying the connection that incurred the significant event.

ConEventHandler is called as an AST routine. If **ConEventHandler** is omitted, then no connection event handler is called when a significant event occurs.

EventFlag

type:	longword
access:	read only
mechanism:	by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If **EventFlag** is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type:	procedure entry mask
access:	call without stack unwinding
mechanism:	by reference

The address of a routine to be called when the connection is established. **ASTHandler** is called as normal AST routine and is passed the parameter **ASTParam** by value. If **ASTParam** is omitted, the value zero is passed instead.

ASTParam

type:	longword
access:	read only
mechanism:	by value

The parameter to be passed to **ASTHandler** when the connection is established. **ASTParam** is passed by value to **ASTHandler** and may be used to identify the connection that has just become established (or for any other compatible purpose). If **ASTParam** is omitted, the value zero is passed to **ASTHandler**.

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry that has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATK$DSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__CONTABFUL	The connection table is full.
ATK__CONLISACT	A connection listener is active on the specified socket.
ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.
ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYS$CLREF

LIB$GET_VM

ATK$DDP_OPN_SKT

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__CONOPNFAI	The connection opening failed.

ATK__CONDENIED	The connection was denied by the remote connection end
ATK__UNAREMADR	Unacceptable address for the remote connection end.

ATK$DSP_CLEAR_COUNTERS – Clear the ADSP internal counters

ATK$DSP_CLEAR_COUNTERS clears ADSP's internal counters. These counters describe the internal significant events that ADSP has incurred.

Format

```
ATK$DSP_CLEAR_COUNTERS     Refnum
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end whose counters will be cleared. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is not specified, or is specified as zero, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.

ATK$DSP_CLOSE – Close a connection

ATK$DSP_CLOSE terminates all connection I/O and frees all resources allocated by a connection end.

Format

```
ATK$DSP_CLOSE    Refnum [,Wait] [,CompStatus] [,EventFlag]
                 [,ASTHandler] [,ASTParam]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end to be closed. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATKDSP_ACCEPT, ATKDSP_ESTABLISH, or ATK$DSP_CREATE_CON_END. If Refnum is not specified, or is specified as zero, an error is returned.

Wait

type: longword
access: read only
mechanism: by value

Specifies whether the connection is to be closed after all data bytes have been successfully received by the remote connection end, or closed immediately. A zero value for Wait forces ADSP to close the connection immediately, without waiting for the send or send attention message queues to empty. Setting Wait to a nonzero value will delay closing the connection until the send queues are empty. If Wait is not specified, ADSP will not wait for the send queues to empty.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to close the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection actually closes. The event flag is cleared immediately and set once the connection goes to the closed state. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by value

The address of a routine to be called when the connection actually closes. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection actually closes. ASTParam is passed by value to ASTHandler and may be used to identify the connection end which has just closed (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

SYS$CLREF

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.

ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_CREATE_CON_END – Create a connection end

ATK$DSP_CREATE_CON_END creates a connection end that will later be used to establish a connection (via ATK$DSP_ESTABLISH).

Format

```
ATK$DSP_CREATE_CON_END        Refnum,Socket,LocalCID
```

Arguments

Refnum

type: longword
access: write only
mechanism: by reference

The connection reference number. Refnum identifies the connection end for all future operations on the connection.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the value of Socket is zero, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. The connection socket number will be returned. If the Socket argument is omitted, an error is returned.

LocalCID

type: word
access: write only
mechanism: by reference

The 16-bit identifier to be used for the local connection end. LocalCID uniquely distinguishes the connection end from other connection ends that may be attached to the same socket.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__CONTABFUL	The connection table is full.
ATK__CONLISACT	A connection listener is active on the specified socket.
ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYS$CLREF

LIB$GET_VM

ATK$DDP_OPN_SKT

ATK$DSP_CREATE_LIS_SOC – Create a connection-listening socket

ATK$DSP_CREATE_LIS_SOC creates a socket that can be used by a connection listener.

Format

```
ATK$DSP_CREATE_LIS_SOC        Socket
```

Arguments

Socket

type: byte
access: modify
mechanism: by reference

The socket number to use to create a connection-listening socket. If specified as zero, ADSP will request DDP to create a socket for the connection listener and return the number of the created socket. If Socket is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.

Any conditions returned by

LIB$GET_VM

ATK$DDP_OPN_SKT

ATK$DSP_DELETE_LIS_SOC – Delete a connection-listening socket

ATK$DSP_DELETE_LIS_SOC terminates socket I/O operations and frees all resources associated with the socket.

Format

```
ATK$DSP_DELETE_LIS_SOC      Socket
```

Arguments

Socket

type: byte
access: read only
mechanism: by value

The socket number of the connection-listening socket that will be deleted. All connection-listening operations are terminated on the socket. If Socket is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__INVSKTNUM	Invalid DDP socket number.
ATK__NOTCONLIS	Not a connection listening socket.

ATK$DSP_DENY_CON_REQ – Deny a connection request

ATK$DSP_DENY_CON_REQ sends a connection denial message to the remote end.

Format

```
ATK$DSP_DENY_CON_REQ Socket,RemoteCID,RemoteAddress
```

Arguments

Socket

type: byte
access: read only
mechanism: by value

The DDP socket number through which to send the connection denial message. Socket must identify a socket that is currently open, otherwise, an error is returned. If Socket is not specified, or is specified as zero, an error is returned.

RemoteCID

type: word
access: read only
mechanism: by value

The 16-bit identifier of the connection end that is the target of the connection denial message. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket. If RemoteCID is not specified, or is specified as zero, an error is returned.

RemoteAddress

type: unspecified
access: read only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the connection denial message. The first two bytes contain the 16 bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. If RemoteAdress is not specified, or is specified as zero, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__INVSKTNUM	Invalid DDP socket number.
ATK__INVCONID	Invalid connection identifier.
Any conditions returned by LIB$GET_VM	

ATK$DSP_ESTABLISH – Establish a connection with a remote end

ATK$DSP_ESTABLISH bypasses the internal ADSP connection-opening dialog and forces a connection end to the open state.

Format

```
ATK$DSP_ESTABLISH Refnum,RemoteAddress,RemoteCID,SendSeq,
                  RecvWdw,RecvSeq,AttnSendSeq,AttnRecvSeq,
                  [,ConEventHandler] [,OPDefaults]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The connection reference number. Refnum must be the reference number of a connection end created by ATK$DSP_CREATE_CON_END; otherwise an error will be returned.

RemoteAddress

type: unspecified
access: read only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

RemoteCID

type: word
access: read only
mechanism: by value

The 16-bit identifier of the connection end that is the target of the Open request. RemoteCID uniquely distinguishes the target connection end from other connection ends that may be attached to the same socket.

SendSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next untransmitted data byte in the local connection end's send queue. If SendSeq is omitted, an error is returned.

RecvWdw

type: word
access: read only
mechanism: by value

The size of remote connection end's receive queue. If it is omitted, an error is returned.

RecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next data byte the local connection end will accept into its receive queue. If it is omitted, an error is returned.

AttnSendSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message which will be transmitted from the local connection end. If it is omitted, an error is returned.

AttnRecvSeq

type: longword
access: read only
mechanism: by value

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format `ConEventHandler SigEvent,Refnum`

Arguments `SigEvent`

type: longword
access: read only
mechanism: by value

The type of significant event that has occurred. `SigEvent` will have one of four values: `ATK__ATTMSGRCV`, which indicates the receipt of an attention message; `ATK__FORRESRCV`, which indicates the receipt of a forward reset; `ATK__CLOADVRCV`, which indicates the receipt of a close advice from the remote end; and `ATK__REMENDUNR`, which indicates the connection being torn down due to inability to reach the remote end.

`Refnum`

type: longword
access: read only
mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

`ConEventHandler` is called as an AST routine. If `ConEventHandler` is omitted, then no connection event handler is called when a significant event occurs.

`OPDefaults`

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If `OPDefaults` is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to `ATK$DSP_SET_DEFAULTS`). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__INVCONSTA	Invalid connection state.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.

ATK$DSP_FLUSH – Flush the send queue

ATK$DSP_FLUSH transmits all data bytes remaining in the send queue.

Format

```
ATK$DSP_FLUSH    Refnum
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end whose send queue is to be flushed. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_FORWARD_RESET – Forward-reset the connection

ATK$DSP_FORWARD_RESET performs a forward reset on the connection.

Format

```
ATK$DSP_FORWARD_RESET  Refnum,CompStatus [,EventFlag]
                       [,ASTHandler] [,ASTParam]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end that will be forward-reset. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to forward-reset the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection has been forward-reset. The event flag is cleared immediately and set once the forward reset has been accepted by the remote ADSP. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection has been forward-reset. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection has been forward-reset. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just been forward-reset (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__OPEALRPRO	The operation is already in progress.

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_GET_BUF – Get a transmission buffer

ATK$DSP_GET_BUF gets a buffer from the memory pool, to be used for data transmission across the connection.

Format

```
ATK$DSP_GET_BUF      Bufptr, BufLength
```

Arguments

Bufptr

type: longword
access: write only
mechanism: by reference

A longword that stores the address of the allocated buffer. If the specified amount of memory cannot be allocated, Bufptr remains unchanged.

BufLength

type: longword
access: read only
mechanism: by value

The length in bytes of the buffer to allocate from the memory pool.

Conditions

SS$_NORMAL Normal successful completion.

Any conditions returned by

LIB$GET_VM

ATK$DSP_GET_BYTE – Get a single byte

ATK$DSP_GET_BYTE reads a single data byte from the receive queue. If the receive queue is empty, an error is returned.

Format

```
ATK$DSP_GET_BYTE Refnum,Byte
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which to obtain the byte. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Byte

type: byte
access: write only
mechanism: by reference

The value of the next data byte available from the receive queue.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__FWDRSTPNT	The forward reset point was reached.
ATK__LOGENDMES	A logical end of message byte was encountered.
ATK__RCVQUEEMP	The receive queue is empty.

ATK$DSP_GET_CON_REQ – Get a connection request

ATK$DSP_GET_CON_REQ gets the next connection request delivered to the listening socket.

Format

```
ATK$DSP_GET_CON_REQ  Socket,RemoteAddress,RemoteCID,
                     PktNextRecvSeq,PktRecvWdw,
                     PktAttnRecvSeq,CompStatus [,EventFlag]
                     [,ASTHandler] [,ASTParam]
```

Arguments

Socket

type: byte
access: read only
mechanism: by value

The socket number of the connection listening socket. If it is omitted, an error is returned.

RemoteAddress

type: unspecified
access: write only
mechanism: by reference

The 4-byte AppleTalk internet address of the connection end that issued the open connection request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. If RemoteAdress is omitted, an error is returned.

RemoteCID

type: word
access: write only
mechanism: by reference

The 16-bit identifier which uniquely identifies the connection end that issued the open connection request. If it is not specified, an error is returned.

PktNextRecvSeq

type: longword
access: write only
mechanism: by reference

The sequence number of the next data byte the remote connection end will accept into its receive queue. If it is not specified, an error is returned.

PktRecvWdw

type: word
access: write only
mechanism: by reference

The size in bytes of the remote connection end's receive window. If it is omitted, an error is returned.

PktAttnRecvSeq

type: longword
access: write only
mechanism: by reference

The sequence number of the next attention message the remote connection end will accept. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to get a connection request. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes. If `CompStatus` is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when a connection request is received. The event flag is cleared immediately and set once a connection request is received. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection request is received. `ASTHandler` is called as a normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection request is received. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just received an open request (or for any other client-specified purpose). If ASTParam is not specified, the value zero is passed to ASTHandler.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__INVSKTNUM	Invalid DDP socket number.
ATK__NOTCONLIS	Not a connection listening socket.
ATK__OPEALRPRO	The operation is already in progress.

Any conditions returned by

SYS$CLREF

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_GET_COUNTERS – Get ADSP internal counters

ATK$DSP_GET_COUNTERS obtains a list of internal counters maintained by ADSP. These counters describe the internal significant events which ADSP has incurred.

Format

```
ATK$DSP_GET_COUNTERS    Refnum,CountList
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which to obtain the counters. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

CountList

type: longword
access: write only
mechanism: by reference

The list of counters that will be returned in the form of a tagged list. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK$K_CntLen bytes in length. The first byte of each entry is the tag, identifying the ADSP internal counter the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP internal counter follows the length field in the entry. If the CountList argument is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.

ATK$DSP_GET_DEFAULTS – Get ADSP default operating parameters

ATK$DSP_GET_DEFAULTS obtains the current or baseline ADSP default operating parameters.

Format

```
ATK$DSP_GET_DEFAULTS   OPDefaults [,Baseline]
```

Arguments

OPDefaults

type: unspecified
access: write only
mechanism: by reference

A tagged list of the operating parameters ADSP is currently using. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK$K_DefLen bytes in length. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP operating parameter follows the length field in the entry. If OPDefaults is omitted, an error is returned. See Appendix A for a list of operating parameters.

Baseline

type: longword
access: read only
mechanism: by value

A flag indicating whether the baseline or current default operating parameters are required. If the Baseline argument is nonzero, the baseline operating parameters are returned. If the Baseline argument is zero, or is not specified, the current operating defaults are returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__PARLISINV	The requested parameter list is invalid.

ATK$DSP_GET_STATUS - Get connection status

ATK$DSP_GET_STATUS obtains the current status of the specified connection end.

Format

```
ATK$DSP_GET_STATUS    Refnum,ConStatus
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which the status will be obtained. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

ConStatus

type: unspecified
access: write only
mechanism: by reference

A tagged list of parameters describing the current state of the connection and the network management counters for the connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list must be at least ATK$K_StaLen bytes in length. The first byte of each entry is the tag, identifying the contents of the entry. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or length field. The value of the entry follows the length field. If ConStatus is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.

ATK$DSP_LISTEN – Passively open a connection

ATK$DSP_LISTEN creates a connection end, then waits for an incoming connect request from a remote socket.

Format

```
ATK$DSP_LISTEN  Refnum,Socket,CompStatus [,RemoteAddress]
                [,FilterList] [,ConEventHandler] [,EventFlag]
                [,ASTHandler] [,ASTParam]  [,OPDefaults]
```

Arguments

Refnum

type: longword
access: write only
mechanism: by reference

The connection reference number. Refnum identifies the connection end for all future operations on the connection. The value of Refnum is available immediately upon return from this call, regardless of whether the call was issued synchronously.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the value of Socket is zero, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of Socket is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for Socket. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. The connection socket number will be returned. If the Socket argument is omitted, an error is returned. The value of Socket is available immediately upon return from this call, regardless of whether the call was issued synchronously.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to establish the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

RemoteAddress

type: unspecified
access: write only
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

FilterList

type: unspecified
access: read only
mechanism: by reference

The list of acceptable internet addresses with which ADSP may establish a connection. Each element of the list is a 4-byte quantity; the first two bytes contain an acceptable network address, the third byte contains an acceptable node number, and the fourth byte an acceptable socket number. A zero may be used as a wildcard character in either the network, node, or socket fields. The effect of a wildcard is to match any number in the field. Therefore, an entry with the network, node, and socket fields all equal to zero will match any internet address. An entry with the network field equal to three, the node field equal to five, and the socket field set to zero will match any socket on node number five in network number three. The list is scanned from the top down (which will affect which internet addresses are accepted) and is terminated with an entry with the network, node, and socket fields set to -1. If FilterList is omitted, ADSP will establish a connection with any internet address.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format `ConEventHandler SigEvent,Refnum`

Arguments `SigEvent`

type:	longword
access:	read only
mechanism:	by value

The type of significant event that has occurred. `SigEvent` will have one of four values: `ATK__ATTMSGRCV`, which indicates the receipt of an attention message; `ATK__FORRESRCV`, which indicates the receipt of a forward reset; `ATK__CLOADVRCV`, which indicates the receipt of a close advice from the remote end; and `ATK__REMENDUNR`, which indicates the connection being torn down due to inability to reach the remote end.

`Refnum`

type:	longword
access:	read only
mechanism:	by value

The connection reference number identifying the connection that incurred the significant event.

`ConEventHandler` is called as an AST routine. If `ConEventHandler` is omitted, then no connection event handler is called when a significant event occurs.

`EventFlag`

type:	longword
access:	read only
mechanism:	by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

`ASTHandler`

type:	procedure entry mask
access:	call without stack unwinding
mechanism:	by reference

The address of a routine to be called when the connection is established. `ASTHandler` is called as normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection is established. ASTParam is passed by value to ASTHandler and may be used to identify the connection which has just become established (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry The length is the size of the entry field in bytes, not including the tag field or length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATK$DSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__CONTABFUL	The connection table is full.
ATK__CONLISACT	A connection listener is active on the specified socket.
ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.
ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYS$CLREF

LIB$GET_VM

ATK$DDP_OPN_SKT

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_OPEN – Actively open a connection

ATK$DSP_OPEN actively seeks to open a connection with the specified socket.

Format

```
ATK$DSP_OPEN    Refnum,RemoteAddress,CompStatus [,Socket]
                [,FilterList] [,ConEventHandler] [,EventFlag]
                [,ASTHandler] [,ASTParam] [,OPDefaults]
```

Arguments

Refnum

type: longword
access: write only
mechanism: by reference

The connection reference number. Refnum identifies the connection end for all future operations on the connection. Refnum returns its value immediately upon the submission of the open call, enabling the client to use this information prior to the completion of the routine. If the open call is made synchronously, it then completes synchronously and returns all completion values at once.

RemoteAddress

type: unspecified
access: modify
mechanism: by reference

The 4-byte AppleTalk internet address of the socket that is the target of the Open request. The first two bytes contain the 16-bit network address. The first byte is the least significant byte, followed by the most significant byte in the second byte. The third byte contains the node number, and the fourth contains the socket number. The address of the socket where the connection is actually made will be returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to establish the connection. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes.

Socket

type: byte
access: modify
mechanism: by reference

The DDP socket number through which the connection is to be established. If the address of `Socket` is specified as zero or is not supplied, ADSP will request DDP to open the next available dynamic socket for the connection end. If the value of `Socket` is zero, ADSP will open a dynamic socket for the connection end. If the value of `Socket` is nonzero, ADSP will first check to see if the socket is already open. If the socket is already open it must be owned by ADSP; otherwise an error is returned. If the socket is not open, ADSP will open it. Well-known sockets may be opened at any time, whereas dynamic sockets must first be opened by ADSP by specifying a value of zero for `Socket`. The newly opened socket can then be used in future ADSP open requests to open multiple connections on the same socket. If the address of the `Socket` argument is supplied, and is nonzero, the connection socket number will be returned. The `Socket` argument returns its value immediately upon the submission of the `open` call, enabling the client to use this information prior to the completion of the routine. If the `open` call is made synchronously, it then completes synchronously and returns all completion values at once.

FilterList

type: unspecified
access: read only
mechanism: by reference

The list of acceptable internet addresses with which ADSP may establish a connection. Each element of the list is a 4-byte quantity; the first two bytes contain an acceptable network address, the third byte contains an acceptable node number, and the fourth byte an acceptable socket number. A zero may be used as a wildcard character in either the network, node, or socket fields. The effect of a wildcard is to match any number in the field. Therefore, an entry with the network, node, and socket fields all equal to zero will match any internet address. An entry with the network field equal to three, the node field equal to five, and the socket field set to zero will match any socket on node number five in network number three. The list is scanned from the top down (which will affect which internet addresses are accepted) and is terminated with an entry with the network, node, and socket fields set to -1. If `FilterList` is omitted, ADSP will establish a connection with any internet address.

ConEventHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of an AST routine to be called whenever a significant event occurs on the connection. The significant events that can occur are the receipt of an attention message, the receipt of a forward reset, the closing of the connection by the remote end, or the connection being torn down when the remote end becomes unreachable. The routine is called with two parameters: the first parameter identifies the significant event; the second identifies the connection end.

This routine is called with the following format:

Format `ConEventHandler SigEvent,Refnum`

Arguments `SigEvent`

type: longword
access: read only
mechanism: by value

The type of significant event that has occurred. `SigEvent` will have one of four values: `ATK__ATTMSGRCV`, which indicates the receipt of an attention message; `ATK__FORRESRCV`, which indicates the receipt of a forward reset; `ATK__CLOADVRCV`, which indicates the receipt of a close advice from the remote end; and `ATK__REMENDUNR`, which indicates the connection being torn down due to inability to reach the remote end.

`Refnum`

type: longword
access: read only
mechanism: by value

The connection reference number identifying the connection that incurred the significant event.

`ConEventHandler` is called as an AST routine. If `ConEventHandler` is omitted, then no connection event handler is called when a significant event occurs.

`EventFlag`

type: longword
access: read only
mechanism: by value

The event flag number to set when the connection is established. The event flag is cleared immediately and set once the connection goes to the open state. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

`ASTHandler`

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the connection is established. `ASTHandler` is called as normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the connection is established. ASTParam is passed by value to ASTHandler and may be used to identify the connection which has just become established (or for any other client-specified purpose). If ASTParam is not specified, the value zero is passed to ASTHandler.

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use for this connection. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry which has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry field in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, the current default operating parameter list is used (the current default operating parameter list is specified with a call to ATK$DSP_SET_DEFAULTS). If there is no current default list, the baseline operating parameters are used. See Appendix A for a list of operating parameters.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__CONTABFUL	The connection table is full.
ATK__CONLISACT	A connection listener is active on the specified socket.
ATK__NOTSKTOWN	ADSP is not the owner of the specified DDP socket.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.
ATK__INVSKTNUM	Invalid socket number.

Any conditions returned by

SYS$CLREF

LIB$GET_VM

ATK$DDP_OPN_SKT

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__CONOPNFAI	The connection opening failed.
ATK__CONDENIED	The connection was denied by the remote connection end.
ATK__UNAREMADR	Unacceptable address for the remote connection end.

ATK$DSP_PUT_BYTE – Send a single byte

ATK$DSP_PUT_BYTE writes a single data byte to the send queue for transmission to the remote end.

Format

```
ATK$DSP_PUT_BYTE Refnum,Byte
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end to accept the byte for transmission to the remote end. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Byte

type: byte
access: read only
mechanism: by value

The value of the data byte to be inserted into the data stream. If it is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

LIB$GET_VM

ATK$DSP_READ – Read data from a connection

ATK$DSP_READ gets a buffer of data bytes received from the remote connection end.

Format

```
ATK$DSP_READ    Refnum,Bufptr,BufLength,Dataptr,ByteCount,
                CompStatus [,EOM] [,EventFlag] [,ASTHandler]
                [,ASTParam]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which to read data. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Bufptr

type: longword
access: write only
mechanism: by reference

A longword which will receive a pointer to the next buffer of data obtained from the connection. If it is omitted, an error is returned.

BufLength

type: longword
access: write only
mechanism: by reference

The length in bytes of the buffer containing the data. If it is omitted, an error is returned.

Dataptr

type: longword
access: write only
mechanism: by reference

A longword which will receive a pointer to where the data actually starts inside the buffer. If it is omitted, an error is returned.

ByteCount

type: longword
access: write only
mechanism: by reference

The number of data bytes in the buffer, starting at `Dataptr`. If it is omitted, an error is returned.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to read data from the connection. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes. If `CompStatus` is omitted, an error is returned.

EOM

type: longword
access: write only
mechanism: by reference

The ADSP logical end-of-message flag. Upon completion of the read, if bit zero is set in EOM, then the last byte in the buffer is the last byte of a logical (client-intelligible) message.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the read completes. The event flag is cleared immediately and set once the connection goes to the open state. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the read completes. `ASTHandler` is called as a normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is not specified, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when the read completes. ASTParam is passed by value to ASTHandler and may be used to identify the read which has just completed (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__FWDRSTPNT	The forward reset point was reached.

Any conditions returned by

SYS$CLREF

LIB$GET_VM

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.
ATK__FWDRSTPNT	The forward reset point was reached.

ATK$DSP_RECV_ATTENTION – Receive an attention message

ATK$DSP_RECV_ATTENTION is used to receive an attention message from the remote connection end.

Format

```
ATK$DSP_RECV_ATTENTION Refnum,Bufptr,BufLength,Dataptr,
                       ByteCount,AttnCode,CompStatus,
                       [,EventFlag] [,ASTHandler] [,ASTParam]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which the attention message will be received. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Bufptr

type: longword
access: write only
mechanism: by reference

A longword to receive a pointer to the buffer containing the next attention message delivered to the connection end. If it is omitted, an error is returned.

BufLength

type: longword
access: write only
mechanism: by reference

The length in bytes of the buffer containing the attention message. If it is omitted, an error is returned.

Dataptr

type: longword
access: write only
mechanism: by reference

A longword which will receive a pointer to where the data actually starts inside the buffer. If it is omitted, an error is returned.

ByteCount

type: longword
access: write only
mechanism: by reference

The number of attention message data bytes contained in the buffer, starting at `Dataptr`. If it is omitted, an error is returned.

AttnCode

type: word
access: write only
mechanism: by reference

The attention code from the ADSP header.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to receive an attention message. `CompStatus` is set to zero initially, then changed to reflect the status of the call when it completes. If it is omitted, an error is returned.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when an attention message has been received. The event flag is cleared immediately and set once the attention message has been received. If `EventFlag` is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when an attention message has been received. `ASTHandler` is called as a normal AST routine and is passed the parameter `ASTParam` by value. If `ASTParam` is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to ASTHandler when an attention message has been received. ASTParam is passed by value to ASTHandler and may be used to identify the connection that has just received an attention message (or for any other client-specified purpose). If ASTParam is omitted, the value zero is passed to ASTHandler.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

SYS$CLREF

Completion status values returned:

SS$_NORMAL	Normal successful completion.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

ATK$DSP_RET_BUF – Return a receive buffer

ATK$DSP_RET_BUF returns a receive buffer obtained from ADSP to the memory pool.

Format

```
ATK$DSP_RET_BUF     Bufptr, BufLength
```

Arguments

Bufptr

type: longword
access: read only
mechanism: by value

A pointer to the buffer being returned to the memory pool.

BufLength

type: longword
access: read only
mechanism: by value

The length in bytes of the buffer being returned to the memory pool.

Conditions

SS$_NORMAL	Normal successful completion.

Any conditions returned by

LIB$FREE_VM

ATK$DSP_SEND_ATTENTION – Send an attention message

ATK$DSP_SEND_ATTENTION sends an attention message to the remote end. Only one attention message can be sent at a time.

Format

```
ATK$DSP_SEND_ATTENTION Refnum,Buffer,BufLength,AttnCode,
                       CompStatus [,ByteCount] [,EventFlag]
                       [,ASTHandler] [,ASTParam]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end from which the attention message will be sent. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Buffer

type: unspecified
access: read only
mechanism: by reference

The buffer that contains the attention message data. It must be the address of the beginning of a buffer obtained through an ATKDSP_GET_BUF, ATKDSP_READ, or ATK$DSP_RECV_ATTENTION call. The buffer becomes the exclusive property of ADSP and cannot be reused by the client. If Buffer is omitted, an error is returned.Note that to send an attention message containing no data, BufLength and ByteCount must be set to zero.

BufLength

type: longword
access: read only
mechanism: by value

The length of Buffer in bytes. This value must be the length returned by a call to ATKDSP_GET_BUF, ATKDSP_READ, or ATK$DSP_RECV_ATTENTION. If it is omitted, an error is returned.

AttnCode

type: word
access: read only
mechanism: by value

The attention code to be placed in the ADSP header. Note that bit 16 of AttnCode is reserved for use by Apple Computer, Inc.

CompStatus

type: longword
access: write only
mechanism: by reference

A longword that receives the results of attempting to send the attention message. CompStatus is set to zero initially, then changed to reflect the status of the call when it completes. If CompStatus is omitted, an error is returned.

ByteCount

type: longword
access: read only
mechanism: by value

The number of attention message data bytes contained in buffer. The value of ByteCount is limited by the maximum attention message size (572 bytes) or BufLength, whichever is smaller. If ByteCount is omitted, the value of Buflength is used instead.

EventFlag

type: longword
access: read only
mechanism: by value

The event flag number to set when the attention message has been sent. The event flag is cleared immediately and set when the attention message has been sent. If EventFlag is not specified, or is specified as zero, no event flag is changed.

ASTHandler

type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

The address of a routine to be called when the attention message has been sent. ASTHandler is called as a normal AST routine and is passed the parameter ASTParam by value. If ASTParam is omitted, the value zero is passed instead.

ASTParam

type: longword
access: read only
mechanism: by value

The parameter to be passed to `ASTHandler` when the attention message has been sent. `ASTParam` is passed by value to `ASTHandler` and may be used to identify the message that has just been sent (or for any other client-specified purpose). If `ASTParam` is omitted, the value zero is passed to `ASTHandler`.

Conditions

`SS$_NORMAL`	Normal successful completion.
`ATK__INSNUMARG`	Insufficient number of arguments supplied.
`ATK__BADREFNUM`	Bad connection reference number.
`ATK__BADPARAM`	Bad parameter.
`ATK__CONNOTOPN`	The connection is not in the open state.
`ATK__CONCLOLOC`	The connection was closed by the local connection end.
`ATK__CONCLOREM`	The connection was closed by the remote connection end.
`ATK__CONTORDOW`	The connection was torn down.
`ATK__OPEALRPRO`	The operation is already in progress.
`ATK__ILLATTCOD`	Illegal attention message code.

Any conditions returned by

`SYS$CLREF`

Completion status values returned:

`SS$_NORMAL`	Normal successful completion.
`ATK__CONCLOLOC`	The connection was closed by the local connection end.
`ATK__CONCLOREM`	The connection was closed by the remote connection end.
`ATK__CONTORDOW`	The connection was torn down.

ATK$DSP_SEND_EOM – Send an end-of-message marker

ATK$DSP_SEND_EOM places an end-of-message marker in the send queue.

Format

```
ATK$DSP_SEND_EOM Refnum
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end into which the EOM will be inserted. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADREFNUM	Bad connection reference number.
ATK__CONNOTOPN	The connection is not in the open state.
ATK__CONCLOLOC	The connection was closed by the local connection end.
ATK__CONCLOREM	The connection was closed by the remote connection end.
ATK__CONTORDOW	The connection was torn down.

Any conditions returned by

LIB$GET_VM

ATK$DSP_SET_DEFAULTS – Set ADSP default operating parameters

ATK$DSP_SET_DEFAULTS sets the default operating parameters for ADSP. (Each future connection uses these operating defaults unless its connection-establishment call explicitly changes the defaults for that specific connection.)

Format

```
ATK$DSP_SET_DEFAULTS   OPDefaults
```

Arguments

OPDefaults

type: unspecified
access: read only
mechanism: by reference

A tagged list of the operating parameters ADSP is to use as the current default operating parameters. Each connection established after calling ATK$DSP_SET_DEFAULTS will use the new default operating parameters for the connection, unless the connection establishment call provides its own set of operating parameters. The list is a series of entries, each of which contains an identifier (the tag), the length of the entry field, and the contents of the field. The list is terminated with an entry that has a zero tag field. The first byte of each entry is the tag. The tag identifies the ADSP operating parameter that the entry contains. The next byte specifies the length of the entry. The length is the size of the entry in bytes, not including the tag field or the length field. The value of the ADSP operating parameter follows the length field in the entry. Each entry in the list is checked for validity against a master list of operating parameters. If OPDefaults is omitted, an error is returned. See Appendix A for a list of operating parameters.

Conditions

SS$_NORMAL	Normal successful completion.
ATK__INSNUMARG	Insufficient number of arguments supplied.
ATK__BADTUNPAR	Bad tuning parameter.
ATK__BADPARLEN	Bad parameter length.

ATK$DSP_WRITE – Write data to a connection

ATK$DSP_WRITE inserts a buffer of data bytes into the send queue for transmission to the remote end.

Format

```
ATK$DSP_WRITE    Refnum,Buffer,BufLength [,ByteCount] [,EOM]
                 [,Flush]
```

Arguments

Refnum

type: longword
access: read only
mechanism: by value

The reference number of the connection end into which the data will be inserted. Refnum must refer to a connection end previously established with ATKDSP_OPEN, ATKDSP_LISTEN, ATK$DSP_ACCEPT, or ATK$DSP_ESTABLISH. If Refnum is omitted, an error is returned.

Buffer

type: unspecified
access: read only
mechanism: by reference

The buffer that contains the data to be inserted into the data stream. It must be the address of the beginning of a buffer obtained through an ATKDSP_GET_BUF, ATKDSP_READ, or ATK$DSP_RECV_ATTENTION call. The specified buffer becomes the exclusive property of ADSP and cannot be reused by the client. If Buffer is omitted when BufLength is nonzero, an error is returned.

BufLength

type: longword
access: read only
mechanism: by value

The length of Buffer in bytes. This value must be the length returned by a call to ATKDSP_GET_BUF, ATKDSP_READ, or ATK$DSP_RECV_ATTENTION. If BufLength is omitted, an error is returned. Zero is an acceptable value for BufLength, indicating that a logical end-of-message (and no data) is being inserted into the data stream, or that the send queue is being flushed. If BufLength is specified as zero, Buffer is not accessed.

ByteCount

type: longword
access: read only
mechanism: by value

The number of bytes contained in the buffer to be inserted into the data stream. This value may be any number from zero to `BufLength`. If it is not specified, or is specified as zero, the value of `Buflength` is used instead.

EOM

type: longword
access: read only
mechanism: by value

The logical end-of-message flag. If nonzero, a logical EOM is inserted into the data stream right after the last byte in the buffer. If EOM is not specified, or is specified as zero, no logical end-of-message is inserted into the data stream.

Flush

type: longword
access: read only
mechanism: by value

If specified as nonzero, all remaining data bytes in the send queue are transmitted to the remote connection end. If `Flush` is not specified, or is specified as zero, data bytes remaining in the send queue are transmitted at the appropriate time.

Conditions

`SS$_NORMAL`	Normal successful completion.
`ATK__INSNUMARG`	Insufficient number of arguments supplied.
`ATK__BADREFNUM`	Bad connection reference number.
`ATK__BADPARAM`	Bad parameter.
`ATK__CONNOTOPN`	The connection is not in the open state.
`ATK__CONCLOLOC`	The connection was closed by the local connection end.
`ATK__CONCLOREM`	The connection was closed by the remote connection end.
`ATK__CONTORDOW`	The connection was torn down.

Any conditions returned by

`LIB$GET_VM`

Appendix A

Tuning Parameter Identifiers

The ADSP for VMS tuning parameter identifiers discussed in this appendix are

```
ATK$K_ATTNBCKOFF
ATK$K_ATTNINTRVL
ATK$K_FWDRSTBCKOFF
ATK$K_FWDRSTINTRVL
ATK$K_OPENINTRVL
ATK$K_OPENMAXRETRY
ATK$K_RCVQCPYTHRESH
ATK$K_RCVQMAXSIZE
ATK$K_RESTARTBCKOFF
ATK$K_RESTARTINTRVL
ATK$K_RETRANSBCKOFF
ATK$K_RETRANSINTRVL
ATK$K_RETRANSTHRESH
ATK$K_SNDQBUFSIZE
ATK$K_SNDWDWCLOBCKOFF
ATK$K_SNDWDWCLOINTRVL
ATK$K_SNDWDWCLOTHRESH
ATK$K_TRANSTHRESH
ATK$K_TRANSTIMERINTRVL
```

ATK$K_ATTNBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The attention-message backoff rate defines how quickly ADSP backs off from retransmitting an unacknowledged attention message. The first retransmission occurs when the attention-message interval expires. The time interval between succeeding retransmissions is governed by the attention-message backoff rate. Each succeeding time interval is multiplied by the attention-message backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between successive retransmissions does not change.

ATK$K_ATTNINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The attention-message retransmit interval defines the time interval between retransmissions of an ADSP attention message. When the client issues a send attention message request, ADSP builds an attention message, transmits it, and starts the attention-message retransmit timer. At each expiration of this timer, the attention message is retransmitted.

ATK$K_FWDRSTBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The forward-reset backoff rate defines how quickly ADSP backs off from retransmitting an unacknowledged forward reset. The first retransmission of the forward reset occurs when the forward-reset interval expires. The time interval between succeeding retransmissions is governed by the forward-reset backoff rate. Each succeeding time interval is multiplied by the forward-reset backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between successive retransmissions does not change.

ATK$K_FWDRSTINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The forward-reset retransmit interval defines how often ADSP retransmits a forward-reset message. When the client issues a forward-reset request, ADSP builds a forward-reset message and transmits it. The forward-reset retransmit timer is also started. At each expiration of this timer, the forward-reset message is retransmitted.

ATK$K_OPENINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The retransmit open request interval defines the time interval between retransmissions of an open connection request. Open connection requests are sent in response to the reception of an open connection request from the remote end, or in response to an active open request from the local client (`ATK$DSP_OPEN` or `ATK$DSP_ACCEPT`). At each expiration of the retransmit open request interval, an open connection request message is transmitted to the remote end.

ATK$K_OPENMAXRETRY

Field Length:	4 bytes	Min:	0
Usage:	unsigned longword	Max:	256
Default:	32		

This parameter defines the number of attempts ADSP makes to establish a connection with a remote end. Each time the retransmit open request interval expires, the open-retry count is incremented. Once this count reaches the value of the maximum retry parameter, the attempt to open the connection is aborted. Using a value of zero for this parameter forces ADSP to continue retransmitting open connection requests until the connection is actually established.

ATK$K_RCVQCPYTHRESH

Field Length:	4 bytes	Min:	20 bytes
Usage:	unsigned longword	Max:	572 bytes
Units:	bytes	Default:	20 bytes

The receive-queue copy threshold determines whether ADSP tries to copy data received from the remote end into the last buffer in the receive queue (to avoid wasted space), or simply links the data into the receive queue. Data received from the remote end is in the form of a DDP datagram. If the amount of data in a received datagram is less than the receive-queue copy threshold, the data is copied into the last buffer in the receive queue; otherwise, the new datagram is linked. If there is no room in the previous buffer, or none exists, the datagram is linked regardless of the value of the receive-queue copy threshold.

ATK$K_RCVQMAXSIZE

Field Length:	4 bytes	Min:	2048 bytes
Usage:	unsigned longword	Max:	65536 bytes
Units:	bytes	Default:	4096 bytes

The receive-queue maximum size defines the original size of the receive window. During connection establishment, the remote end is notified that the local end's receive queue is `ATK$K_RCVQMAXSIZE` bytes in length.

ATK$K_RESTARTBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The restart-sender backoff rate defines how quickly ADSP backs off from transmitting acknowledgment requests in order to restart the remote end's data transmission. The first ACK request is transmitted after the restart-sender interval expires. The time interval between succeeding ACK requests is governed by the restart-sender backoff rate. Each succeeding time interval is multiplied by the restart-sender backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK$K_RESTARTINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The restart-sender interval defines how often ADSP transmits ACK request messages in order to notify the remote end that the receive window is open. When the receive window reopens after becoming closed, an ACK request message is transmitted and the restart-sender timer is started. At each expiration of the restart-sender interval, another ACK request message is transmitted. Once an acknowledgment is received from the remote end, the timer is stopped.

ATK$K_RETRANSBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The retransmit backoff rate defines how quickly ADSP backs off from transmitting ACK requests in order to retransmit data. The first ACK request is transmitted after the retransmit interval expires. The time interval between succeeding ACK requests is governed by the retransmit backoff rate. Each succeeding time interval is multiplied by the retransmit backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK$K_RETRANSINTRVL

Field Length:	4 bytes	Min:	0 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The retransmit interval defines how long ADSP waits for the acknowledgment of transmitted data. Once a datagram is transmitted to the remote end, the retransmit interval timer is started. If an acknowledgment is not received before the interval expires, an ACK request message is transmitted to the remote end, and the timer is restarted. Any acknowledgment received after the original expiration of the timer causes all unacknowledged data to be retransmitted. The unacknowledged data is assumed to be lost. The timer is then restarted at its original interval. The timer is stopped when all outstanding data bytes are acknowledged by the remote end.

ATK$K_RETRANSTHRESH

Field Length:	4 bytes	Min:	0
Usage:	unsigned longword	Max:	10
Units:	datagrams	Default:	4

The retransmit advice threshold defines when the local ADSP implementation sends a retransmit advice message to the remote end. If the count of consecutive out-of-sequence datagrams received by the local end reaches the value of the retransmit advice threshold, a retransmit advice message is transmitted. The reception of an in sequence datagram resets the counter.

ATK$K_SNDQBUFSIZE

Field Length:	4 bytes	Min:	100 bytes
Usage:	unsigned longword	Max:	16384 bytes
Units:	bytes	Default:	112 bytes

The send queue buffer size defines the size of buffers ADSP allocates for the send queue in response to `ATK$DSP_PUT_BYTE` requests. Each Put Byte request issued to ADSP attempts to copy the data byte into the last buffer in the send queue. When this buffer is full, a new buffer is allocated, which is `ATK$K_SNDQBUFSIZE` bytes in size.

ATK$K_SNDWDWCLOBCKOFF

Field Length:	4 bytes	Min:	1
Usage:	unsigned longword	Max:	100
Units:	multiplier	Default:	2

The send window closed backoff rate defines how quickly ADSP backs off from transmitting ACK requests in order to open the send window. The first ACK request is transmitted after the send window closed interval expires. The time interval between succeeding ACK requests is governed by the send window closed backoff rate. Each succeeding time interval is multiplied by the send window closed backoff rate until the connection-probe interval is reached (30 seconds). When this happens, the connection-probe interval is used instead. Using a backoff rate of 1 prevents ADSP from backing off—that is, the time interval between retransmissions does not change.

ATK$K_SNDWDWCLOINTRVL

Field Length:	4 bytes	Min:	0.3 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The send window closed interval defines how often ADSP solicits the remote end for acknowledgment information once the send window closes. Once the send window closes, the send window closed timer is started. At each expiration of the send window closed interval, an ACK request message is transmitted to the remote end. Once an acknowledgment that opens the send window is received, the timer is stopped.

ATK$K_SNDWDWCLOTHRESH

Field Length:	4 bytes	Min:	0 bytes
Usage:	unsigned longword	Max:	8192 bytes
Units:	bytes	Default:	1024 bytes

The send window closing threshold defines the point in the send queue where ADSP will send an ACK request in anticipation of the send window closing. When transmitting data to the remote end, ADSP passes through the send queue until it reaches the point where the send window closes and it cannot send anymore data. At this point, ADSP must wait for the reception of an acknowledgment from the remote end, which opens the send window. The send window closing threshold marks the point in the send queue preceding the send window closed point. When ADSP reaches this point, it sets the ACK request bit in the ADSP header of the next transmitted datagram. This request solicits an acknowledgment from the remote end, which hopefully advances the send window before the local end reaches the send window closed point. An ACK request is always sent immediately when the send window closes.

ATK$K_TRANSTHRESH

Field Length:	4 bytes	Min:	0 bytes
Usage:	unsigned longword	Max:	572 bytes
Units:	bytes	Default:	112 bytes

The transmit threshold is the number of unsent data bytes ADSP accumulates in the send queue before transmitting the data to the remote end. A counter is incremented each time a byte is inserted into the send queue. When this counter reaches the value of the transmit threshold, a datagram is built and the data is transmitted. The counter is then reset to zero.

ATK$K_TRANSTIMERINTRVL

Field Length:	4 bytes	Min:	0 seconds
Usage:	unsigned longword	Max:	30 seconds
Units:	0.1 seconds	Default:	1 second

The transmit timer interval defines how long ADSP waits for the transmit threshold to be reached. The transmit timer is started when the first data byte is inserted into the send queue. If the transmit timer interval expires before the transmit threshold is reached, all unsent data bytes are transmitted to the remote end. The transmit threshold counter is then reset to zero. If the transmit threshold is reached before the interval expires, the timer is either restarted or stopped. If there remains any unsent data in the send queue, the timer is restarted; otherwise it is stopped.

Appendix B

Connection Status Identifiers

Each of the connection status identifiers described below is used as an unsigned longword with a field length of 4 bytes.

Identifier	Description
ATK$K_ATTNMSGWAIT	A logical value indicating whether an attention message is waiting in the receive queue. If set to nonzero, an attention message is waiting. If set to zero, the receive queue is empty.
ATK$K_ATTNRCVSEQ	The sequence number of the next attention message that will be accepted from the remote end.
ATK$K_ATTNSNDSEQ	The sequence number of the next attention message to be transmitted to the remote end.
ATK$K_FIRSTRTMTSEQ	The sequence number of the first unacknowledged byte in the send queue.
ATK$K_RCVSEQ	The sequence number of the next byte that will be accepted from the remote end.
ATK$K_RCVWDW	The current size of the local receive window.
ATK$K_SNDSEQ	The 32-bit sequence number of the first untransmitted byte in the send queue.

Identifier	Description
ATK$K_SNDWDWSEQ	The sequence number of the last byte in the send queue that may be transmitted to the remote end (that is, the send window close point).
ATK$K_STATE	The state of the connection. This field can take on one of the values that follow.
ATK$K_INVALID	The connection control block is invalid.
ATK$K_LISTEN	The connection end is passively waiting for an incoming open connection request.
ATK$K_OPENING	The connection end is actively seeking to open a connection with a remote end.
ATK$K_MYHALFOPEN	The connection end has obtained the state information of the remote end, and is actively seeking to open the connection.
ATK$K_HISHALFOPEN	The remote connection end has obtained the local end's state information and the local end is actively seeking to open the connection.
ATK$K_OPEN	The connection is operational.
ATK$K_TORNDOWN	The connection is half-open since the remote end is unreachable.
ATK$K_CLOSING	The local client has issued a close request, and the connection is waiting for the send queues to empty.
ATK$K_CLOSED	A close advice has been received from the remote end, and the connection end is waiting for the receive queues to empty.

Appendix C

ADSP Internal Counter Identifiers

Each of the internal counter identifiers described below is used as an unsigned longword with a field length of 4 bytes.

Identifier	Description
ATK$K_ATTNACKRCVD	The number of attention message acknowledgments received from the remote end.
ATK$K_ATTNACKACPTD	The number of attention message acknowledgments accepted by the local end.
ATK$K_PROBERCVD	The number of probe messages received from the remote end.
ATK$K_ACKRCVD	The number of explicit ACK messages received from the remote end.
ATK$K_FWDRSTRCVD	The number of forward-reset messages received from the remote end.
ATK$K_FWDRSTACPTD	The number of forward-reset messages accepted by the local end.
ATK$K_FWDRSTACKRCVD	The number of forward-reset acknowledgment messages received from the remote end.
ATK$K_FWDRSTACKACPTD	The number of forward-reset acknowledgments accepted by the local end.

Identifier	Description
ATK$K_ATTNRCVD	The number of attention messages received from the remote end.
ATK$K_ATTNACPTD	The number of attention messages accepted by the local end.
ATK$K_DATARCVD	The number of data messages received from the remote end.
ATK$K_DATAACPTD	The number of data messages accepted by the local end.
ATK$K_ACKFIELDCHKD	The number of times the acknowledgment fields of the ADSP header were examined.
ATK$K_ACKNRSFIELDACPTD	The number of times the Next Receive Sequence Number field was accepted by the local end.
ATK$K_ACKSWSFIELDACPTD	The number of times the calculated Send Window Sequence Number field was accepted by the local end.
ATK$K_ACKREQSTD	The number of times an acknowledgment was requested by the remote end.
ATK$K_LOWMEM	The number of times the local end encountered low memory conditions.
ATK$K_OPNREQEXP	The number of times the open request timer expired.
ATK$K_PROBEEXP	The number of times the probe timer expired.
ATK$K_FWDRSTEXP	The number of times the forward-reset retransmission timer expired.
ATK$K_ATTNEXP	The number of times the attention-message retransmission timer expired.
ATK$K_TRANSEXP	The number of times the transmit interval timer expired.
ATK$K_RETRANSEXP	The number of times the retransmit unacknowledged data timer expired.
ATK$K_SNDWDWCLOEXP	The number of times the send window closed timer expired.
ATK$K_RESTARTEXP	The number of times the restart-sender timer expired.
ATK$K_RESLOWEXP	The number of times the resources-low timer expired.
ATK$K_RETRANSRCVD	The number of times a retransmit advice message was received.